Titre: Title:	Forecasting rail transportation demand using artificial neural networks
Auteur: Author:	Shadi Sharif Azadeh
Date:	2007
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Sharif Azadeh, S. (2007). Forecasting rail transportation demand using artificial neural networks [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8024/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/8024/
Directeurs de recherche: Advisors:	
Programme: Program:	Non spécifié

UNIVERSITÉ DE MONTRÉAL

FORECASTING RAIL TRANSPORTATION DEMAND USING ARTIFICIAL NEURAL NETWORKS

SHADI SHARIF AZADEH

DÉPARTEMENT DE MATHÉMATIQUES
ET DE GÉNIE INDUSTREL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(MATHÉMATIQUES APPLIQUÉES)
AOÛT 2007

Your file Votre référence
ISBN: 978-0-494-35700-2
Our file Notre référence
ISBN: 978-0-494-35700-2

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

FORECASTING RAIL TRANSPORTATION DEMAND USING ARTIFICIAL NEURAL NETWORKS

présenté par : SHARIF AZADEH, Shadi

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M . ADJENGUE Luc, Ph.D., président

M .LABIB Richard, Ph.D., membre et directeur

M . SAVARD Gilles, Ph.D., membre et codirecteur de recherche

M . MARCOTTE Patrice, Ph.D., membre

To my dear parents
and my beloved Yousef
for their unconditional love and support

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisors, Professor Gilles Savard and Richard Labib for accepting me in their research group, for their confidence in my abilities and for their encouragements during my work which made difficulties much easier during the last two years. I am also grateful to Dr. Jean-Philippe Côté, for his help, kindness and his great availability during my project.

I would like to thank my parents, my parents in-law, my brother, my dear husband for their patience, unconditional love, and encouragements. This work was not possible without their presence. Moreover, I would like to appreciate my friends Sanaz, Khadija and Mounira and Dr. Raffie's family for their support.

RÉSUMÉ

Cette étude analyse le potentiel d'un réseau de neurones artificiel pour prédire les réservations et les annulations de billets, d'une classe de prix donnée, dans différentes gares données, lors un jour donné, pour une importante compagnie ferroviaire. Puisque la réservation est traitée plusieurs semaines avant le départ, il y a toujours une possibilité que la réservation soit annulée. Il est donc important de faire des prévisions fiables des réservations et des annulations avant le départ.

Le modèle utilisé est Le Perceptron Multi-Couches (MLP). Ce réseau modélise la relation entre les variables dépendantes et indépendantes qui sont tirées des historiques des réservations et des annulations. Après la capacité de généralisation du réseau de neurones est examinée. L'ensemble des données proviennent de la compagnie de transport européenne TGV pour l'itinéraire Paris-Bruxelles.

Un choix judicieux de prétraitement des données a été effectué pour accélérer l'apprentissage. Un réseau de neurones supervisé, avec deux couches cachées, chacune formée de cinq neurones, est entraîné en utilisant un algorithme d'apprentissage de rétro-propagation. Les résultats numériques sont concluants et démontrent une bonne généralisation du réseau de neurones, puisque l'erreur de prédiction est inférieure à 8%.

ABSTRACT

This study analyzes the use of an artificial neural network to produce accurate forecasts of total bookings and cancellations, in a particular fare class on a specific date, at various points before departing of a major rail operator. Since booking is processed several weeks before departure, there is always a possibility that a reservation could be canceled. It is therefore important to forecast both bookings and cancellations before departure.

The model used is a Multi-Layer Perceptron (MLP) describing the relationship between independent and dependent values based on historical reservations and cancellations. The generalization ability of the network is tested to evaluate the predicting accuracy of the regression model for future trends of reservations and cancellations using actual railroad data. The dataset is provided by TGV transportation in Europe, i.e. the Paris-Brussels itinerary.

Relevant pre-processing approaches have been employed to make learning more efficient. A supervised neural network, with two hidden layers each comprised of five neurons, is trained using a back-propagation learning algorithm. Numerical results show good generalization since the predictions are within 8% of the targeted values.

CONDENSÉ EN FRANÇAIS

Le transport joue un rôle central dans l'économie, mais variation des bénéfices et les coûts des transports sont souvent sujets à de grandes incertitudes. On peut voir l'importance de la prédiction de demande dans les stratégies d'organisation de transport. Puisque le transport doit répondre rapidement aux changements, particulièrement sur une grande échelle, il y a une forte demande de données plus opportunes et les décisions ne peuvent pas attendre des mesures à être inventées. Des techniques statistiques de prédiction pour des activités comme la tarification, le contrôle des stocks, la prédiction de demande et la surréservation, sont largement utilisées dans les industries du transport.

Des approches conventionnelles afin d'étudier la prédiction de la demande ont été étudiées pendant des années, et elles supposent que les utilisateurs prennent des décisions basés sur la maximisation d'une fonction d'utilité. Ces approches incluent les séries temporelles, les modèles régressifs, le lissage exponentiel, et les méthodes bayésiennes pour évaluer les paramètres des modèles prédictifs. Bien que ces méthodes soient énormément appliquées dans la prédiction de la demande, elles ont quelques inconvénients qui nous motivent à tourner notre attention vers de nouvelles méthodes. Récemment, quelques méthodes de représentation ont été appliquées à la modélisation en transport; ces méthodes améliorent ou remplacent les approches analytiques existantes.

Des méthodes de prédiction sont utilisées pour améliorer les processus de décision. Les obstacles dans les problèmes de décision sont de pouvoir répondre à des questions comme : quelle variable prédire, avec quelle méthode, quel pas de temps, et quel degré de précision est désiré. Ces questions sont encore plus compliquées dans un contexte de transport. L'exactitude de prévision exigée a un impact sur le système de prédiction

adopté. Un des obstacles supplémentaires est la disponibilité de données historiques, qui est primodial dans l'établissement de procédures de prédiction.

Les méthodes prédictives peuvent être qualitatives ou quantitatives, selon les méthodes mathématiques et statistiques utilisées. Des procédures qualitatives impliquent l'évaluation subjective par des avis d'experts. Il y a quelques procédures formelles utilisées dans ce type de prédiction, comme le marketing d'essais, des enquêtes clients, des évaluations de force de vente et des données historiques, mais le processus pour obtenir la prévision est subjectif. D'autre part, des approches statistiques définissent explicitement comment la prévision est décidée mathématiquement, ce qui est exigé dans un contexte de transport. Ces méthodes examinent des données historiques pour vérifier le processus sous-jacent produisant la variable à l'étude et supposent que le processus est stable.

La sélection de méthodes de prédiction dépend de plusieurs facteurs, comme le format de prévision exigée, la disponibilité de données, la précision exigée et la facilité de mise en opération. Il y a les nombreuses méthodes de prédiction de transport, qui peuvent être divisées en quatre catégories principales.

La première approche d'intérêt est appelée série temporelle Selon la définition de Montgomery , le processus est mis en équation de sorte à pouvoir être prolongé dans l'avenir [1]. Le modèle n'a pas besoin de présenter de très vieilles observations vu qu'ils n'expriment pas le comportement actuel. Aussitôt qu'un modèle de séries temporelles applicable est obtenu, une technique de prédiction appropriée peut être développée [1]. Cette méthode est largement utilisée dans des contextes de prédiction de transport. Malgré les capacités de l'approche de séries temporelles, ce modèle ne peut pas répondre rapidement aux changements soudains de réservations et d'annulations.

La deuxième approche est l'analyse de régression, une technique statistique pour modéliser et examiner la relation entre deux variables ou plus. En 1992, Rengaraju et Arasan ont examiné la prédiction de la demande du voyage aérien en utilisant une analyse de régression linéaire multiple [9]. En 2004, Tianjin utilisa des modèles de régression dans la prédiction de demande de vol [6] et, en 2006, Anderson a appliqué des régressions linéaires multiples pour diriger la prédiction de demande de volumes de trafic dans des petites communautés urbaines [7]. Cependant, pour un grand ensemble de données avec de nombreux facteurs de prédictions et des variables de réponse avec plusieurs observations, les calculs peuvent être très compliqués et longues.

Une autre approche de prévision est le lissage exponentiel. C'est probablement la classe de procédures la plus répandue pour le lissage de séries temporelles discrètes afin de prédire l'avenir immédiat. Sa popularité peut être attribuée à sa simplicité, à l'efficacité informatique et à l'exactitude raisonnable; cependant, cette méthode a quelques inconvénients. Il y a une constante dans cette méthode appelée *constante de lissage*, qui est ajusté par une quantité proportionnelle à l'erreur de prévision la plus récente. Selon la définition de Montgomery, si cette constante est relativement petite, on donne plus de poids aux données historiques. D'autre part, si elle est grande, plus de poids est accordé à l'observation actuelle [1]. Un inconvénient majeur du lissage exponentiel est qu'il est difficile de choisir une valeur optimale pour la constante sans faire des suppositions restrictives du comportement de la série temporelle.

L'approche finale appliquée pour évaluer les paramètres est la méthode bayésienne. En 1997, Popovic a examiné une méthode adaptative pour produire des entrées de demande aux modèles de contrôle des stocks de siège de compagnie aérienne. Les données statistiques ne sont pas souvent disponibles pour pouvoir les utiliser comme une base pour prédire la demande future de passagers. Popovic utilise la méthode bayésienne pour développer une approche afin de traiter avec le manque d'informations historiques [15]. En 2007, Miltenburg He a utilisé des informations de demande dans un processus

d'évaluation bayésien pour réviser les prévisions de demande [14]. Cependant, bien que cette méthode travaille précisément pour définir les paramètres d'un modèle de régression, il y a toujours le problème de déterminer la distribution de données historiques.

L'utilisation de ces méthodes dans le contexte des transports exige toujours les évaluations de beaucoup de paramètres et coefficients pour prévoir des tendances futures. Donc, le nombre d'observations historiques doit être augmenté, ce qui, pour des ensembles de données non-linéaires complexes, complique l'obtention de résultats raisonnables. Les séries temporelles ne peuvent pas répondre rapidement aux changements soudains, qui sont très courants dans la structure de données en transport. De plus, ces méthodes requièrent de faire certaines suppositions sur la distribution de données historiques.

Pour surmonter certains des inconvénients des méthodes conventionnelles, nous tournons notre attention vers une nouvelle méthode qui travaille plus précisément dans des espaces non-linéaires : les Réseaux de Neurones Artificiels (RNA). La sortie dans des modèles de régression est la somme linéaire des réponses pondérées, tandis que dans des réseaux de neurones, des combinaisons linéaires multiples sont traitées en parallèle, c'est-à-dire l'activation dans chaque neurone est une combinaison linéaire séparée. L'avantage majeur de l'approche de réseau neuronale consiste en sa flexibilité pour modéliser des rapports non-linéaires de complexité arbitraire d'une façon automatisée [17]. La propriété la plus intéressante des réseaux neuronaux multicouches non récurrents est leur capacité de se rapprocher aussi précisément que désiré à une fonction des exemples de l'ensemble d'apprentissage. En fait, un réseau neuronal non récurrent à trois couches entièrement connecté avec n entrées, un nombre suffisant nœuds cachés et un nœud de sortie, peut être formé pour approximer une fonction de configuration $n \times 1$ de complexité arbitraire [17]. Pour obtenir les informations exigées sur chaque itinéraire, un intervalle de temps spécifique est choisi et dans la prédiction de la

demande future, beaucoup d'informations manquantes doivent être traitées. Rakes (1991) a noté que les réseaux neuronaux exécutent bien avec les données manquantes ou incomplètes, une tâche qui est difficile pour d'autres techniques corrélationnelles. Zahedi (1991) a déclaré que des réseaux neuronaux ont l'avantage supplémentaire de succès d'exécution là où d'autres méthodes échouent souvent à reconnaître et trouver une courbe de tendance aux systèmes compliqués, vagues, ou incomplets [18]. Normalement, la méthode de Rétro Propagation d'Avance Directe (RPAD) est employée pour former les réseaux neuronaux. Comme Hashemi a observé en 1995, la performance de réseaux neuronaux RPAD est plus efficace que des méthodes statistiques et stochastiques conventionnelles dans la prédiction continue de série de temps [19].

Inspirés par le système neuronal biologique, les RNA sont des réseaux de traitement distribués parallèlement qui sont modelés à l'image des structures corticales du cerveau. Ils sont constitués d'éléments de traitement interconnectés, appelés des nœuds ou des neurones, qui travaillent ensembles pour produire une fonction de sortie [17]. La sortie de réseaux neuronaux compte sur la coopération des neurones individuels dans le réseau. Les réseaux neuronaux n'éliminent pas les techniques statistiques ou n'automatisent pas complètement la tâche de prédiction; au lieu de cela, ils sont destinés pour aider les évaluateurs dans la fabrication de rajustements systématiques et cohérents à la prédiction de demande produites par des techniques statistiques [20].

Un perceptron, l'équivalent artificiel du neurone, est un classificateur linéaire qui est le composant structurel de base d'un réseau. Le perceptron est un classificateur linéaire qui est capable de dessiner une ligne entre deux régions séparables. Le processus de formation commence par l'assignation d'une ligne selon des valeurs de poids initiales et le processus de formation continue tant que la ligne n'est pas placée en un endroit juste; c'est-à-dire là où l'erreur est aussi faible que possible.

Maintenant que nous avons présenté la structure d'un neurone simple, nous sommes intéressés par la considération de la structure plus compliquée de perceptron multicouche. Dans des réseaux neuronaux plus complexes et particulièrement dans les perceptrons multicouches (PMC), les échantillons de formation (x_i, d_i) entrent au réseau par le processus d'étude. Les entrées $\{x_i\}$ ont un impact sur les variables de réponse $\{d_i\}$. Pendant le processus d'étude, le réseau alloue quelques coefficients, ou des poids, aux variables d'entrée $\{x_i\}$. Les valeurs optimales des poids sont ajustées en réduisant au minimum la différence entre la fonction "réelle" $f(x)$ et la fonction "sortie de réseau". C'est tout comme l'observation d'un phénomène stochastique qui peut être décrit par un vecteur aléatoire $\vec{X}$, qui consiste de *variables indépendantes* et des *variables dépendantes* $\vec{D}$. De plus, les N réalisations du vecteur aléatoire $\vec{X}$ sont dénotées par $\{x_i\}_{i=1}^{N}$ et les valeurs correspondantes sont $\{d_i\}_{i=1}^{N}$.

Bien que le réseau neuronal soit une des représentations les plus simples et les plus puissantes des solutions mathématiques, obtenir les poids est une tâche difficile d'optimisation non-linéaire et itérative. Le but est de minimiser l'erreur quadratique sur les sorties des cas de formation. La rétro propagation est une des méthodes d'optimisation les plus simples, avec beaucoup de quantités descriptibles incluant la simplicité et l'incohérence de cas de formation; cependant, elle est lente, particulièrement dans des espaces de grandes dimensions. Pour cette méthode d'opération, l'algorithme suit un cycle de cinq étapes pour chaque échantillon de formation $\{(x(n), d(n))\}_{n=1}^{N}$.

La première étape est l'initialisation. En supposant qu'aucune information antérieure n'est disponible, l'algorithme d'apprentissage par rétro-propagation choisit aléatoirement des poids initiaux selon une distribution uniforme. La deuxième étape présente les exemples de formation. Pour chaque exemple dans le jeu, il exécute l'ordre

de calculs d'un sens et de l'autre à travers chaque époque. La troisième est le calcul avant. Considérons un exemple de formation dans l'époque noté par $(x(n), d(n))$, avec le vecteur d'entrée $x(n)$ appliqué à la couche d'entrée et la réponse $d(n)$ (réelle) désirée présenté à la couche de sortie des nœuds de calcul. Calculant le champ local produit et les signaux de fonction du réseau en passant du sens avant par le réseau, couche par couche. Le champ local incité $v_j^{(l)}(n)$ pour le neurone j dans la couche l est

$$v_j^{(l)}(n) = \sum_{i=0}^{m_0} w_{ji}^{(l)}(n) y_i^{(l-1)}(n) \tag{1}$$

où $y_i^{(l-1)}(n)$ est le signal de sortie de neurone dans la couche précédente $l-1$ à l'itération n et $w_{ji}^{(l)}(n)$ est le poids synaptique de neurone dans la couche qui est alimentée du neurone i dans la couche $l-1$. Pour $i=0$, nous avons $y_0^{(l-1)}(n) = +1$ et $w_{j0}^{(l)}(n) = b_j^{(l)}(n)$ et le déplacement appliqué au neurone j dans la couche l est

$$y_j^{(l)} = \varphi(v_j(n)) \tag{2}$$

Si le neurone j est dans la première couche cachée (i.e. $l=1$)., alors

$$y_j^{(0)}(n) = x_j(n) \tag{3}$$

où $x_j(n)$ est l'élément j^{th} du vecteur d'entrée $x(n)$. Si le neurone j est dans la couche de sortie (i.e. $l=L$, où est mentionnée comme la profondeur du réseau), alors

$$y_j^{(L)} = o_j(n) \tag{4}$$

le signal d'erreur est

$$e_j(n) = d_j(n) - o_j(n) \tag{5}$$

où $d_j(n)$ est l'élément j^{th} du vecteur $d(n)$ de réponse (réel) désirée.

La quatrième étape est le calcul en sens inverse. Dans cette étape l'algorithme d'étude calcule les δ du réseau, que nous avons décrit précédemment. Finalement, la dernière étape est l'itération, qui réitère les calculs d'un sens et de l'autre en présentant les

nouvelles époques d'exemples de formation au réseau jusqu'à ce que le critère d'arrêt soit rencontré.

L'utilisation de l'algorithme d'apprentissage par rétro propagation a quelques avantages. Un perceptron multicouche formé avec cet algorithme agit en tant qu'*approximateur universel*. L'intuition suggère qu'un PMC avec des fonctions d'activation douce doit avoir les dérivées de fonction de sortie qui estiment les dérivées d'une configuration d'entrée-sortie. En fait, les PMC peuvent approximer les fonctions qui ne sont pas différentiables dans le sens classique, mais possèdent des dérivées généralisées, comme dans le cas de fonctions différentiables par morceaux. Les résultats d'approximation fournissent une justification théorique précédemment manquante pour l'utilisation des PMC dans les applications qui exigent l'approximation d'une fonction et ses dérivées.

Dans le cas considéré ici, le but est de prévoir le nombre de passagers pour une société de transport ferroviaire majeure. Parce que la différence entre le nombre de réservations et des annulations au temps de départ représente le nombre de passagers, nous avons choisi ces deux facteurs comme sortie du modèle.

Beaucoup de facteurs affectent le nombre de réservations et d'annulations et beaucoup d'informations sont disponibles sur les passagers pour chaque temps de départ. Comme les entrées du réseau, nous utilisons sept facteurs dans le modèle de régression comme prédicteurs; aussi il y a deux variables de réponse. Le processus de réservation commence 120 jours avant le départ et il y a 20 segmentations d'intervalles de réservation. Pendant ces 120 jours avant le départ, les passagers enregistrent les informations pour la réservation d'itinéraire; certaines des réservations pourraient être annulées pendant ce temps. Le jour de départ et le temps de départ ne sont même pas codés quand ils sont des données naïves. Nous codons ces informations en divisant les semaines dans deux groupes et divisant chaque jour en trois catégories principales; (matin, soirée et nuit). Chaque passager appartient à un des 19 types de produits et 14

types de classes physiques. Les réservations sont affectées aussi par des prix différents; comme, le prix de classe, le prix d'itinéraire et le prix moyen de jour. La demande pour la classe affaires en raison de son prix est normalement moindre que la classe économique; aussi le temps du jour fait que le prix de chaque itinéraire est différent des autres. Le jour dans l'année joue un rôle important dans le prix moyen de chaque jour et par conséquent la demande correspondante est différente. Leur combinaison fournit une variété de possibilités et finalement une grande quantité d'informations en entrée du réseau. Selon la fréquence de données et la proportion de points aberrants qui sont négligeables nous les enlevons pour améliorer la performance du réseau.

Comme quatre des sept facteurs qui sont les variables d'entrée du modèle de régression sont codées et le reste des entrées sont des nombres réels, nous appliquons ces données au réseau telles quelles. Nous avons appliqué les données de janvier 2005 pour former et évaluer le réseau. Pour nous assurer de la capacité de généralisation du modèle nous avons utilisé les données de mars 2005. Ainsi il y a deux ensembles de données différents qui sont employés dans le modèle.

Nous divisons les données dans quatre intervalles. La première plage indique le cas où les nombres de réservations ainsi que d'annulations sont tous les deux nulles. La deuxième plage représente le nombre de réservations et d'annulations supérieur à zéro et inférieur à cent et les deux dernières plages montrent le nombre de réservations et d'annulations pour plus de 100 passagers. Nous avons décidé d'avoir quatre segmentations parce que nous visons à éliminer les valeurs aberrantes. La plupart de données sont inférieures à 100 donc la définition de quatre intervalles distincts semble suffisant. La première colonne spécifie le nombre de réservations et la deuxième le nombre d'annulations. Nous enlevons les valeurs aberrantes empiriquement pour améliorer la performance du réseau. Au premier coup d'œil, nous voyons que les valeurs de sortie sont toujours positives, puisque le nombre de réservations et des annulations (i.e. le nombre des réservations qui sont annulées près du jour de départ) est toujours

égal ou plus grand que zéro. Donc, nous pouvons utiliser les distributions qui fournissent des valeurs positives. De plus, la forme des données diminue strictement en termes de chaque intervalle, par exemple, il y a beaucoup de données au point zéro et comme l'échantillon approche 100 (en éliminant les annexes), le nombre de réservations et d'annulations diminuent. Cela suggère de choisir la distribution exponentielle pour dresser la carte des valeurs de sortie dans l'intervalle [0, 1), qui est réversible et nous calculons l'inverse en appliquant la distribution de logarithme.

Le choix du nombre de couches cachées et des neurones cachés est fait empiriquement et il n'y a aucune règle spécifique pour le faire. Une question pratique qui surgit dans ce contexte est celui de minimiser la taille du réseau en gardant la bonne performance. Un réseau neuronaux de taille minimale est peu enfin à introduire du bruit dans les données de formation et peut aboutir à une meilleure généralisation. Nous pourrions utiliser une de deux méthodes : croissance de réseau ou élagage de réseau. Dans cette étude, nous avons choisi la méthode de croissance de réseau, dans le cas où nous commençons par un petit PMC et ajoutons ensuite un nouveau neurone ou une nouvelle couche de neurones cachés. Nous avons appliqué des techniques de momentum et de *régularisation* pour améliorer la performance de réseau. Ces méthodes ont diminué l'erreur significativement. Afin de déterminer le taux d'apprentissage et modifier le processus d'apprentissage, nous avons employé la méthode d'*étude adaptative*. La performance de l'algorithme de descente suivant le gradient de plus grande pente peut être améliorée si nous permettons au taux d'apprentissage de changer pendant le processus de formation. Un taux d'étude adaptatif essayera de garder la taille d'étape d'étude autant que possible en gardant l'apprentissage stable.

D'abord, les paramètres du réseau sont fixés pendant le processus de formation en appliquant une proportion des données qui appartiennent à janvier 2005. Les paramètres ajustés sont examinés par le reste des mêmes données. Afin de vérifier la capacité du réseau dans la généralisation nous avons choisi deux méthodes de *validation mutuelle*.

Le premier est la méthode de K-fold et pour le deuxième, nous avons appliqué un nouvel ensemble de données appartenant à mars 2005.

Le réseau consiste en une couche d'entrée avec sept nœuds, deux couches cachées avec cinq neurones chacun et une couche de sortie avec deux nœuds représentant les réservations et les annulations. Les résultats montrent que l'application de la méthode PMC proposée peut améliorer la prédiction du nombre moyen des passagers avec une erreur de généralisation de 8 %.

.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

Transportation plays a central role in the economy, but there is often great uncertainty about the range of transportation benefits and costs. The importance of detail in demand forecasting can be seen in transportation organization strategies. Because transportation must respond rapidly to changes, especially on a large scale, there is a high demand for more timely data and decisions cannot wait for measurements to be devised. Statistical forecasting techniques for activities such as pricing, inventory control, demand forecasting, and overbooking, are widely used in transportation industries.

Conventional approaches to demand forecasting have been studied for years and they assume that users make decisions based on utility maximization. These approaches include time series, regression, exponential smoothing and the Bayesian method of parameter estimation in a forecasting model. Although these methods are vastly applied in demand forecasting, they have some drawbacks that motivate us to turn our attention to new methods. Recently, some representation methods have been applied in transportation modeling that improve or replace existing analytical approaches.

The artificial neural network (ANN) approach, a non-linear black box model, is a useful alternative for modeling travel demand in the presence of complex non-linear relationships. ANNs are currently being used intensively in diverse engineering applications and have become a regular method to provide solutions to regression and estimation problems. Despite their simple structure, ANNs have the ability to mimic human characteristics of problem solving via learning and generalization. Among various kinds of ANNs, the Multi-Layered Perceptron (MLP) has become the most widely used network architecture in neural network application. It is a special type of

feed-forward back-propagation (FFBP) neural network models that can be improved using for example, regularization, momentum and adaptive learning techniques.

In this research, we investigate demand forecasting of a major railroad. To do so, we apply a three-layer MLP with a back-propagation learning algorithm. Passenger information, such as time of ticket purchase, date of departure, price, product and physical class, is used. Before feeding the network, we treat the incoming information to allow for better learning. In the first (training) phase, we enter samples $\{(x_i, d_i)\}$ into the network in order to determine numerous parameters; afterwards in the second phase we test the adjusted parameters. The aim is to predict the number of reservations (bookings) and cancellations and, consequently, the number of passengers at the time of departure. The architecture proposed consists of two parts: a pre-processing phase, which comprises normalization and a regression phase. To evaluate the generalization ability of the network, we use two methods of cross validation: first, the k-fold method and, second, cross validation with a large dataset. We employ improvement methods to overcome some of the back-propagation drawbacks that affect the generalization (i.e. the ability of the network to precisely predict the response variables for new datasets) error of the network. The network consists of one input layer with seven nodes, two hidden layers with five neurons each and an output layer with two nodes responding bookings and cancellations. The results show that applying the proposed MLP method can improve prediction of the average number of passengers with an 8% generalization error.

In the next chapter we review some conventional methods that are still used in transportation demand forecasting and their drawbacks and we explain our motivation to apply neural networks to these problems. We introduce Artificial Neural Networks as powerful tools in regression problems. In the third chapter we investigate the applied MLP more closely. We start with the data treatment procedure, including outlier elimination and normalization. Then we explain the details of the model, such as the

back-propagation learning algorithm, the method of optimization and further improvements. In Chapter four, we present the results, starting with the determination of the number of hidden neurons and layers. We also compare the results of the network before and after improvements. Finally, we conclude the research and discuss the perspective of this project in Chapter five.

CHAPTER 2 FORECASTING MODELS AND NEURAL NETWORKS

In this chapter, we present statistical methods used in forecasting systems to predict values of variables that are important in decision processes. These methods analyze historical data to provide estimates of the future. In Section 2.1 we consider conventional methods used in demand forecasting; in Section 2.2 we investigate neural networks, which we hope may overcome some of the drawbacks of the conventional methods.

2.1 Statistical approaches used in demand forecasting

Forecasting methods are used to improve the decision process. Obstacles in decision problems deal with answering questions about what is to be forecast, which method is chosen, what time elements are involved and what degree of accuracy is desired. The required forecast accuracy impacts the forecasting system adopted. One of the additional obstacles is the availability of historical data, which is valuable in establishing forecasting procedures.

Forecasting methods can be qualitative or quantitative, depending upon the extent to which mathematical and statistical methods are used. Qualitative procedures involve subjective estimation through the opinions of experts. There are some formal procedures used in this type of prediction, such as marketing tests, customer investigations, sales force estimates and historical data, but the process to obtain the forecast is subjective. On the other hand, statistical approaches explicitly define how the forecast is determined mathematically. These methods examine historical data to verify the underlying process generating the variable under consideration.

The selection of forecasting methods depends on several factors, such as the format forecast required, the availability of data, the accuracy required and the ease of operation. There are numerous methods of prediction in transportation forecasting, which can be divided into four main categories:

1) Time Series Model
2) Regression Model
3) Exponential Smoothing Method
4) Bayesian Method

The first approach of interest is referred to as time series models. According to Montgomery's definition [1], the process is mathematically modeled in such a way that can be extended into the future. The model does not need to present very old observations because the data too far in the past does not express current behavior. As soon as an applicable time series model is obtained, an appropriate forecasting technique can be developed [1]. This method is widely used in transportation forecasting contexts. In 1985, Sen [5] examined air travel demand using time series data, and, in 2002, Devoto and Lilliu [3] investigated the potential of an econometrics approach for forecasting air transport demand. The resulting models were tested on three airports in Sardinia (Cagliari, Olbia, and Alghero) to identify the most suitable and characteristic variables for representing air transport demand. Time series for annual passenger movements, expressed in absolute terms and index numbers, were constructed for each airport [3]. Chung [4] in 2002 applied time series prediction models to automobile demand. He constructed a structural equation model to estimate aggregated automobile demand with data from Korea. Zhang [2] 2006 studied an inventory control problem with time-varying demand. Despite the capabilities of the time series approach in transportation forecasting, these models cannot rapidly respond to sudden changes in bookings and cancellations.

The second approach is regression analysis, a statistical technique for modeling and investigating the relationship between two or more variables. In 1992, Rengaraju and Arasan [9] investigated air travel demand forecasting using a multiple linear regression analysis. In 2004, Tianjin [6] used regression models in flight demand forecasting, and, in 2006, Anderson [7] applied multiple linear regressions in demand forecasting of traffic volumes in small urban communities. Also in 2006, Varagouli et al. [8] applied the multiple regression method to travel demand forecasting for the prefecture of Xanthi in northern Greece. In the transportation context, there are many examples of regression models searching to define the underlying model of one or more input variables that affect the response variable with a set of coefficients. These models for simple low dimensional data work well, as shown in Lin et al. in 2005 [10]. However, for a large dataset with numerous predictors and response variables with many observations, the computations can be very complicated and inverse matrix as well as normal equation calculations make the process time consuming.

Another approach to generate forecast is exponential smoothing. This is probably the most widely used class of procedures for smoothing discrete time series in order to forecast the immediate future and its popularity can be attributed to its simplicity, computational efficiency and reasonable accuracy; however, this method has some drawbacks. There is a constant in this method called the *smoothing constant*, which is adjusted by an amount proportional to the most recent forecast error. According to Montgomery's definition [1], if this constant is relatively small, more weight is given to the historical data. On the other hand, if it is large, more weight is placed on the current observation. In 1999, Snyder et al. [13] applied exponential smoothing methods to inventory control forecasting. In particular, they considered the problem of finding the prediction distribution of aggregate lead-time demand for use in inventory control calculations. In 2000, Godfrey proposed [11] a class of exponential smoothing–based methods that handle the multiple calendar effect and use adaptive estimation of daily demands for freight transportation. In 2006, Widiarta et al. [12] applied this method to

forecast the demand of an item that belongs to a product family. A major drawback of exponential smoothing is that it is difficult to select an optimum value for the constant without making restrictive assumptions about the behavior of the time series. This problem is compounded when the form of the underlying time series models is changing. Several techniques have been developed to monitor and modify automatically the value of the constant. These techniques are called adaptive-control smoothing methods because the smoothing parameter modifies or adapts itself to changes in the underlying time series. The decision rules employed in these adaptive-control methods are quite simple. In the next chapter we will discuss the adaptive-learning method in neural networks, which greatly improves forecasting results.

The final approach applied to estimate parameters is the Bayesian method. In 1997, Popovic [15] investigated an adaptive method for generating demand inputs to airline seat inventory control models. Statistical data are often not available to use as a basis for forecasting future passenger demand. Popovic [15] uses the Bayesian method to develop an approach to deal with the lack of historical information. In 2007, Miltenburg [14] used demand information in a Bayesian estimation process to revise demand forecasts. However, although this method works accurately to define the parameters of a regression model, there is still the problem of determining the distribution of historical data.

Using these methods in the transportation context still requires estimates of many parameters and coefficients to predict future trends. Therefore, the number of historical observations must be increased, which, for complex nonlinear datasets, makes it complicated to obtain reasonable results. Time series cannot respond rapidly to sudden changes, which are very common in transportation data structure. Moreover, these methods require making certain assumptions about the distribution of historical data.

To overcome some of the drawbacks of conventional methods, we turn our attention to a method that works more precisely in nonlinear spaces: Artificial Neural Networks

(ANN). The output in regression models is the linear sum of the weighted responses, whereas in neural networks, multiple linear combinations are processed in parallel, that is, the activation in each neuron is a separate linear combination. The major advantage of the neural network approach is that it is flexible enough to model complex non-linear relationships of arbitrary complexity in an automated fashion [17]. The most valuable property of multilayer feed-forward neural networks is their ability to approximate as accurately as desired a function from training examples. In fact, a three-layer, fully connected feed-forward neural network with n input nodes, a sufficiently large number of hidden nodes and one output node, can be trained to approximate an $n\times1$ mapping function of arbitrary complexity [17]. To obtain the required information about each itinerary, a specific time interval is selected and in forecasting future demand, a lot of missing information must be dealt with. Zahedi [18] in 1991 stated that neural networks have the added advantage of performing successfully where other methods often fail to recognize and match complicated, vague, or incomplete patterns. Normally, the Feed-Forward Back-Propagation method (FFBP) is employed to train the neural networks. As Hashemi [19] concluded in 1995, the performance of FFBP neural networks is more efficient than conventional statistical and stochastic methods in continuous time series forecasting. In Section 2.2 we will investigate the mathematical origin of neural networks and their fundamentals. More details will be considered in Chapter 3.

2.2 Neural networks forecasting model

Inspired by the biological neural system, ANNs are parallel distributed processing networks that are modeled after cortical structures of the brain. They consist of interconnected processing elements, called nodes or neurons, that work together to produce an output function [17]. The output of neural networks relies on the cooperation of the individual neurons within the network. Neural networks do not eliminate statistical techniques or completely automate the forecasting task; instead, they are intended to aid managers in making systematic and consistent adjustments to demand

forecasts produced by statistical techniques [20]. Figure 2-1 shows a biological neuron, the principal component of the biological neural system.

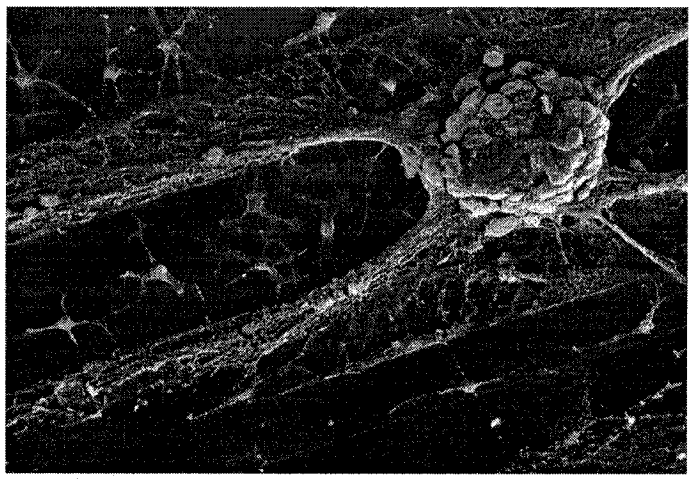

Figure 2-1: Biological Neuron

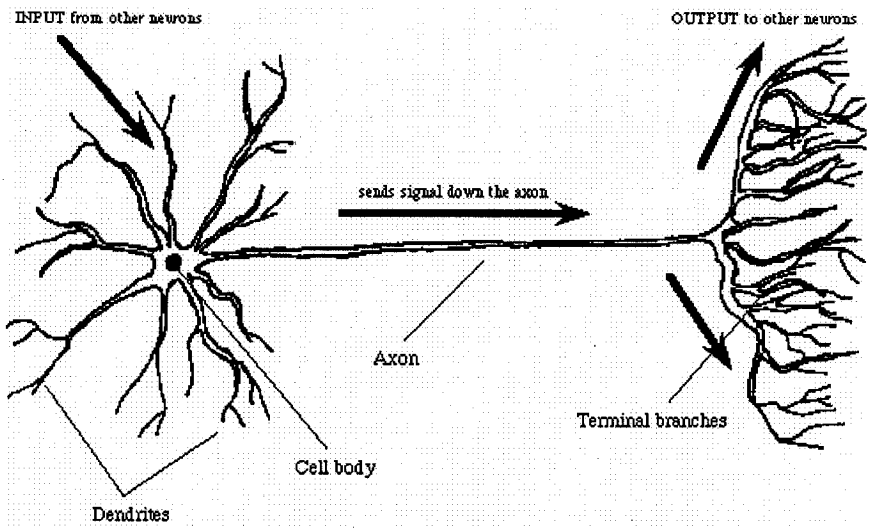

Figure 2-2: Biological Neuron

A perceptron, the artificial equivalent of the neuron, is a linear classifier (i.e. clearly classifies the inputs that belong to only two classes) that is the inherent structural component of a network. Figure 2-2 shows the cortical structure of a biological neuron.

Figure 2-3 illustrates an artificial neuron and its fundamentals. The synaptic weights (i.e. coefficients of the inputs) of the perceptron are denoted by $w_1, w_2, ..., w_m$ and the corresponding inputs are $x_1, x_2, ..., x_m$. The bias is determined by b. The role of the bias is to shift the boundary from the origin.

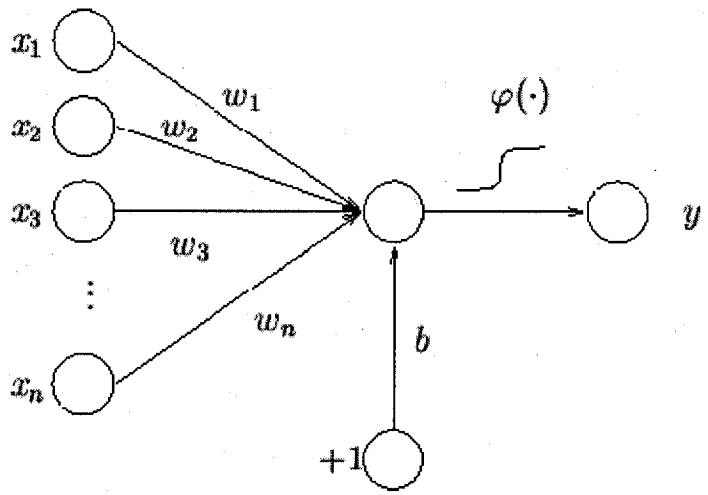

Figure 2-3: Artificial Neuron

The induced local field is determined as it is given by:

$$\upsilon = \sum_{i=1}^{m} w_i x_i + b \tag{2.1}$$

Neural Networks are used in both forecasting and classification problems. Here we consider the classification nature of the percepton. The goal of the perceptron is to correctly classify the inputs $x_1, x_2, ..., x_m$ that belong to either class ℓ_1 or class ℓ_2. In fact, x_i will be assigned to class ℓ_1 if the output is +1 and to class ℓ_2 if the output is -1. These two classes are separated by a *hyperplane* whose equation is (2.1) equals to zero:

$$\sum_{i=1}^{m} w_i x_i + b = 0 \tag{2.2}$$

The synaptic weights $w_1, w_2, ..., w_m$ of the perceptron can be adapted using an iteration-by-iteration process. The bias is treated as a synaptic weight for a fixed input equal to +1. The input vector is defined as follows:

$$x(n) = [+1, x_1(n), x_2(n), ..., x_m(n)]^T \tag{2.3}$$

where n is the iteration step in applying the algorithm. The corresponding weight vector is:

$$w(n) = [b(n), w_1(n), w_2(n), ..., w_m(n)] \tag{2.4}$$

Now, equation (2.1) can be rewritten as

$$v(n) = \sum_{i=0}^{m} w_i(n) x_i(n) = w^T(n) x(n) \tag{2.5}$$

where $w_0(n)$ represents the bias $b(n)$. For a perceptron to be exact, the two classes, ℓ_1 and ℓ_2, must be linearly separable. Let H_1 be the subset of training vectors (i.e. the vectors used to train the network and to fix weights) that belongs to class ℓ_1 and H_2 be the subset of training vectors that belong to class ℓ_2. The union of H_1 and H_2 is the training set H. The training process adjusts the weight vector in a way that the two classes ℓ_1 and ℓ_2 are linearly separable in order to find the weight vector that satisfies following inequalities:

$w^T x > 0$ for every input vector that belongs to class ℓ_1 (2.6a)

$w^T x \leq 0$ for every input vector that belongs to class ℓ_2 (2.6b)

In the iteration process of the perceptron according to the error-correction rule, two different situations may occur. First, $x(n)$ may be correctly classified by the weight vector $w(n)$ computed in the n^{th} iteration of the algorithm, so that no correction is made to the weight vector of the perceptron in the next iteration:

$$w(n+1)=w(n) \quad \text{if } w^T x(n)>0 \text{ and } x(n) \text{ belongs to class } \ell_1 \tag{2.7a}$$

$$w(n+1)=w(n) \quad \text{if } w^T x(n)\le 0 \text{ and } x(n) \text{ belongs to class } \ell_2 \tag{2.7b}$$

Or, weight updating is implemented as

$$w(n+1)=w(n)-\eta(n)x(n) \quad \text{if } w^T(n)x(n)>0 \text{ and } x(n) \text{ belongs to class } \ell_2 \tag{2.8a}$$

$$w(n+1)=w(n)+\eta(n)x(n) \quad \text{if } w^T(n)x(n)\le 0 \text{ and } x(n) \text{ belongs to class } \ell_1 \tag{2.8b}$$

where the learning rate $\eta(n)$ is a constant that simply controls the weight adjustment in the n^{th} iteration.

As we said before, a perceptron is a linear classifier which is capable to draw one line between two separable regions. The training process starts by assigning one line according to initial weight values and the training process continues as long as the line is located in the right place; that is, the error is as small as possible. Figure 2-4 shows two different regions which are presented by blue and red colors. As it is illustrated the initial random position of the classifier is assigned incorrectly before training, but after training the regions are separated accurately. Perceptron via output (i.e. $Y=\varphi(\upsilon)$) draws a line between these two classes.

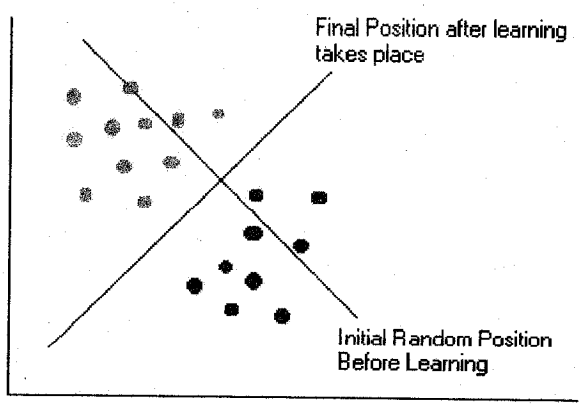

Figure 2-4: Perceptron as a linear classifier

Now that we introduced the structure of a simple neuron we are interested in considering the more complex structure of a multi-layer perceptron. In more complex neural networks and especially in multilayer perceptrons, training samples (x_i, d_i) enter the network through the learning process. Inputs $\{x_i\}$ impact the response variables $\{d_i\}$. During the learning process, the network allocates some coefficients, or weights, to the input variables $\{x_i\}$. Optimal values of the weights are adjusted by minimizing the difference between the "actual" function $f(x)$ (i.e. the distribution that real values follow) and the "network output" function $F(x, w)$. This is the same as the observation of a stochastic phenomenon that can be described by a random vector $\vec{X}$, which consists of *independent variables* and *dependent variables* $\vec{D}$. In addition, N realizations of the random vector $\vec{X}$ are denoted by $\{x_i\}_{i=1}^N$, and the corresponding values are $\{d_i\}_{i=1}^N$. First, it could be useful to introduce the stochastic and statistical nature of the mapping process. Training samples are included in the set

$$\Gamma = \{(x_i, d_i)\}_{i=1}^N \tag{2.9}$$

As the relationship between $\vec{X}$ and $\vec{D}$ is unknown, the general feature of the regression model is

$$D = f(X) + \varepsilon \tag{2.10}$$

Where $f(.)$ is deterministic and ε is a random expectation error representing ignorance about the interdependence of $\vec{X}$ and $\vec{D}$. The mean value of the expected error ε, given any $\vec{X}$, is zero; that is,

$$E(\varepsilon \mid x) = 0 \tag{2.11}$$

The regression function $f(x)$ can be expressed as a conditional mean of the model output D given that the input $X = x$.

$$f(x) = E(D \mid x) \tag{2.12}$$

The expected error ε is uncorrelated with the regression function $f(x)$; thus implies,

$$E(\varepsilon f(x)) = 0 \tag{2.13}$$

The purpose of the model is to use the vector $\vec{X}$ to predict the dependent variables D and encode the empirical knowledge represented by the training sample set Γ into a corresponding set of synaptic weight vectors W. Let the "network output" produced in response to the input vector $\vec{X}$ be denoted by the random variable:

$$Y = F(X, w) \tag{2.14}$$

where $F(., w)$ is the input-output function realized by the network. Given the training dataset Γ, the optimal weights are theoretically obtained by minimizing,

$$\xi(w) = \frac{1}{2}\sum_{i=1}^{N}(d_i - F(x_i, w))^2 \tag{2.15}$$

The cost function $\xi(w)$ is the squared difference between the desired response d and the network outputs y, averaged over the entire training dataset Γ.

Before discussing specific neural network components, we introduce the mathematical algorithm employed to minimize the cost function to adjust the free parameters. To make adjustments on a continuing basis, we use the least-mean-square (LMS) algorithm, which is based on the use of instantaneous values of the cost function; we can rewrite equation (2.15) as

$$\xi(w) = \frac{1}{2} e^2(n) \tag{2.16}$$

where $e(n)$ is the error signal measured at time n. By differentiating $\xi(w)$ with respect to the weight vector w we have

$$\frac{\partial \xi(w)}{\partial w} = e(n) \frac{\partial e(n)}{\partial w} \tag{2.17}$$

The network output is represented as

$$\begin{aligned} y(n) &= x^T(n) w(n) \\ e(n) &= d(n) - y(n) \end{aligned} \tag{2.18}$$

Hence,

$$\frac{\partial e(n)}{\partial w(n)} = -x(n) \tag{2.19}$$

and

$$\frac{\partial \xi(w)}{\partial w(n)} = -x(n) e(n) \tag{2.20}$$

Giving an estimate for the gradient vector

$$\hat{g}(n) = -x(n) e(n) \tag{2.21}$$

In the method of steepest descent, the successive adjustments applied to the weight vector w are in the direction opposite to the gradient vector $g = \nabla \xi(w)$. Accordingly, the steepest descent algorithm is formally described by

$$w(n+1) = w(n) - \eta g(n) \tag{2.22}$$

where η is a positive constant called the *step-size* or *learning-rate parameter*, and $g(n)$ is the gradient vector evaluated at the point $w(n)$. From iteration n to $n+1$, the algorithm applies the correction rule

$$\Delta w(n) = w(n+1) - w(n) = -\eta g(n) \tag{2.23}$$

According to the first-order Taylor series expansion around $w(n)$:

$$\xi(w(n+1)) \cong \xi(w(n)) + g^T \Delta w(n) \tag{2.24}$$

Hence,

$$\xi(w(n+1)) \cong \xi(w(n)) - \eta g^T(n) g(n) \cong \xi(w(n)) - \eta \left\| g(n) \right\|^2 \tag{2.25}$$

For a positive learning rate parameter η, the cost function is decreased from one iteration to the next. This situation is valid as long as a small value for the learning rate is applied. Moreover, this method of steepest descent converges to the optimal solution w^* in a relatively slow fashion. The learning rate parameter η has a profound influence on its convergence behavior. When η is small, weight changing curve follows a path without fluctuations in the W-plane, as can be seen in figure 2-4.

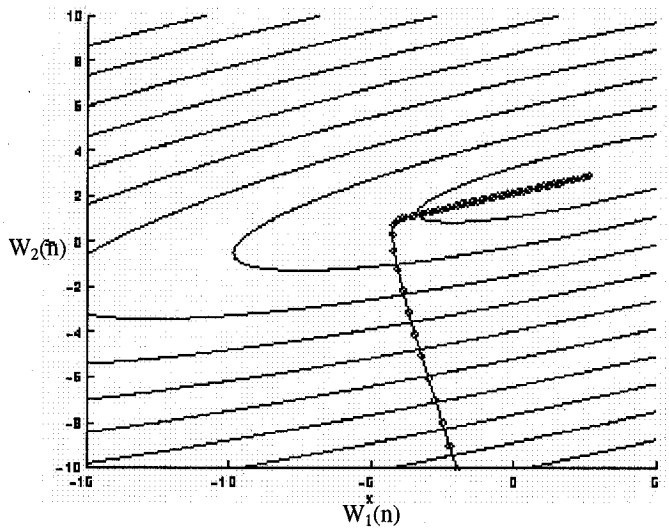

Figure 2-5: Gradient method when η is small

When η is large, the trajectory of weight change follows a zigzagging path and when the learning rate exceeds a certain critical value, the algorithm becomes unstable and it diverges. Figure 2-6 shows the case in which the learning rate is large.

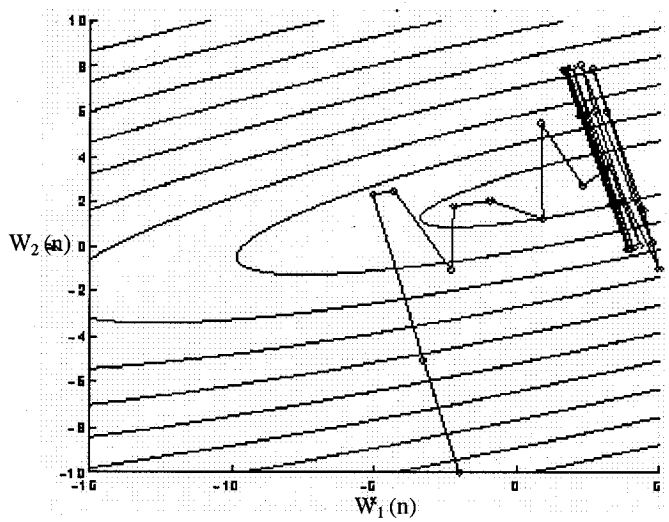

Figure 2-6: Gradient method when η is large

In the LMS algorithm the weight vector $\hat{w}(n)$ traces a trajectory. The LMS is called a stochastic gradient method. It is customary to set the initial values of the weight vector of the algorithm equal to zero.

Although the neural network is one of the simplest and most powerful representations of mathematical solutions, obtaining the weights is a difficult task of nonlinear and iterative optimization. The goal is minimizing the squared error over the outputs of the training cases. Back-propagation is one of the simplest optimization methods, with many describable quantities; however, it is slow, particularly in high dimensions. For this method of operation, the algorithm goes through the training sample $\{(x(n),d(n))\}_{n=1}^{N}$ in five steps.

The first step is initialization. Assuming that no prior information is available, the back-propagation learning algorithm picks uniformly randomized initial weights. The second is the way to present training examples. For each example in the set, it performs the sequence of forward and backward (i.e. when this algorithms sends back the weights through the network to minimize the error function) computations through each epoch (i.e. one complete presentation of the entire training set during the learning process). The third is forward computation. Let a training example in the epoch be denoted by $(x(n),d(n))$, with the input vector $x(n)$ applied to the input layer and the desired (actual) response $d(n)$ presented to the output layer of the computation nodes. Compute the induced local field and function signals of the network by proceeding forward through the network, layer by layer. The induced local field $v_j^{(l)}(n)$ for neuron j in layer l is

$$v_j^{(l)}(n) = \sum_{i=0}^{m_0} w_{ji}^{(l)}(n) y_i^{(l-1)}(n) \tag{2.26}$$

Where $y_i^{(l-1)}(n)$ is the output signal of neuron i in the previous layer $l-1$ at iteration n and $w_{ji}^{(l)}(n)$ is the synaptic weight of neuron j in layer l that is fed from neuron i in

layer $l-1$. For $i=0$, we have $y_0^{(l-1)}(n)=+1$ and $w_{j0}^{(l)}(n)=b_j^{(l)}(n)$, and the bias applied to neuron j in layer l is

$$y_j^{(l)}=\varphi(v_j(n)) \tag{2.27}$$

If neuron j is in the first hidden layer (i.e. $l=1$), set

$$y_j^{(0)}(n)=x_j(n) \tag{2.28}$$

Where $x_j(n)$ is the j^{th} element of the input vector $x(n)$. If neuron j is in the output layer (i.e. $l=L$, where L is referred to as the depth of the network), set

$$y_j^{(L)}=o_j(n) \tag{2.29}$$

Compute the error signal

$$e_j(n)=d_j(n)-o_j(n) \tag{2.30}$$

where $d_j(n)$ is the j^{th} element of the desired (actual) response vector $d(n)$.

The fourth is backward computation and finally, the last step is the iteration, which iterates the forward and backward computations by presenting new epochs of training examples to the network until the stopping criterion is met.

Using a back-propagation learning algorithm has some advantages. A multi-layer perceptron trained with the back-propagation learning algorithm manifests itself as a *universal approximator*. That is, intuition suggests that an MLP with smooth activation functions ($\varphi(v)$) should have output function derivatives that approximate the derivatives of an input-output mapping. In fact, MLPs can approximate functions that are not differentiable in the classical sense, but possess generalized derivatives, as in the case of piecewise differentiable functions. The approximation results provide a previously missing theoretical justification for the use of MLPs in applications that require the approximation of a function and its derivatives.

A learning algorithm is *computationally efficient* when its computational complexity is polynomial in the number of adjustable parameters that are to be updated from one iteration to the next. On this basis, the back-propagation learning algorithm is computationally efficient, especially in using it to train a multi-layer perceptron containing a total of W synaptic weights when its computational complexity is linear in W.

Furthermore, because of the LMS properties, the back-propagation learning algorithm is *robust* and there is less fluctuation in error function. However, this method of training has a low rate of convergence. The error surface in back-propagation is fairly flat along a weight dimension, which means that the derivative of the error surface with respect to that weight is small in magnitude. Moreover, the direction of the negative gradient vector may point away from the minimum of the error surface; hence, the adjustments applied to the weights may induce the algorithm to move in the wrong direction.

We explained briefly the mathematical aspects of the initial components in supervised feed-forward neural networks. Now, we consider the fundamentals of multilayer perceptrons, which are the most popular supervised neural networks. MLPs have been applied successfully to difficult and diverse problems by being trained in a supervised manner using the back-propagation algorithm based on the error correcting learning rule. MLP neural networks are organized in layers: an input layer, one or more hidden layers, and an output layer. The number of neurons along with the number of layers determines the topology of the MLP neural network [19].

An MLP neural network generally contains one or two hidden layers. The number of neurons in the input and output layers is set to match the number of input and target parameters of the process under investigation [16]. The number of neurons in the hidden layers, on the other hand, is determined by experience and some rule of thumb. Therefore, there could be more than one topology for an MLP to model a process successfully. Figure 2-7 presents a Multi-Layers artificial neural network.

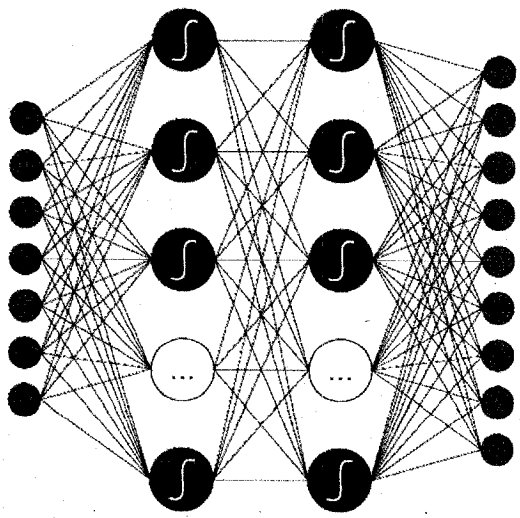

Figure 2-7:MultiLayer Perceptrons

The error signal at the output of neuron j at iteration n for n^{th} training example is defined by

$$e_j(n) = d_j(n) - y_j(n) \tag{2.31}$$

where neuron j is an output node.

The instantaneous value $\xi(n)$ of the total error is obtained by summing $\frac{1}{2}e_j^2(n)$ over all output neurons because these are the only visible neurons that the error signal could be calculated for directly.

$$\xi(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n) \tag{2.32}$$

C includes all the neurons in the output layer.

N is the total number of training examples and the averaged squared error function is obtained by summing $\xi(n)$ over all n and then normalizing with respect to the size N, as shown by

$$\xi_{av} = \frac{1}{N}\sum_{n=1}^{N} \xi(n) \tag{2.33}$$

The instantaneous error functions $\xi(n)$ and, therefore, ξ_{av}, are a function of all the free parameters (synaptic weights and bias levels) of the network. For a given training sample, ξ_{av} represents the cost function as a measure of learning performance. The aim in the learning process is to obtain the optimal values of parameters by minimizing ξ_{av}, for which we use the LMS algorithm.

The output $v_j(n)$ associated with neuron j is

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \tag{2.34}$$

Where m is the total number of inputs, including the bias, applied to neuron j. Therefore, the function signal $y_j(n)$ at the output of neuron j at iteration n is

$$y_j(n) = \varphi_j(\upsilon_j(n)) \tag{2.35}$$

In a manner similar to the LMS algorithm, the back-propagation algorithm applies a correction $\Delta w_{ji}(n)$ to the weight $w_{ji}(n)$, which is proportional to the partial derivative $\frac{\partial \xi(n)}{\partial w_{ji}(n)}$. This derivative may be expressed as follows:

$$\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial \upsilon_j(n)} \frac{\partial \upsilon_j(n)}{\partial w_{ji}(n)} \tag{2.36}$$

Where $\frac{\partial \xi(n)}{\partial w_{ji}(n)}$ indicates the direction of search in weight space for the synaptic weight w_{ji}. The correction

$$\Delta w_{ji}(n) = -\eta \frac{\partial \xi(n)}{\partial w_{ji}(n)} \tag{2.37}$$

where η is the learning-rate parameter of the back-propagation algorithm is applied. The minus is for the gradient descent direction in weight space, we have

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \tag{2.38}$$

Or more specifically,

$$\begin{pmatrix} weight \\ correction \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} learning- \\ rate \\ parameter \\ \eta \end{pmatrix} \cdot \begin{pmatrix} local \\ gradient \\ \delta_j(n) \end{pmatrix} \cdot \begin{pmatrix} input \\ signal \\ of \\ neuron \\ j \\ y_j(n) \end{pmatrix}$$

where the *local gradient* $\delta_j(n)$ is defined as

$$\begin{aligned} \delta_j(n) = -\frac{\partial \xi(n)}{\partial \upsilon_j(n)} &= -\frac{\partial \xi(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial \upsilon_j(n)} \\ &= e_j(n) \varphi_j'(\upsilon_j(n)) \end{aligned} \tag{2.39}$$

The local gradient for the output neuron j is equal to the product of the corresponding error signal $e_j(n)$ for that neuron and the derivative $\varphi_j'(\upsilon_j(n))$ of the associated activation function.

Depending on where neuron j is located, two cases may occur:

1- If neuron j is an output node, $\delta_j(n)$ equals the product of the derivative $\varphi_j'(\upsilon_j(n))$ and the error $e_j(n)$, both of which are associated with neuron j.

2- If neuron j is a hidden node, $\delta_j(n)$ equals the product of the associated derivative $\varphi_j'(\upsilon_j(n))$ and the weighted sum of the δ's computed for the neurons in the next hidden or output layer that are connected to neuron j.

There are advantages and disadvantages of using the back-propagation learning method. There is no specific rule to define the number of hidden neurons and hidden layers, making the selection of number of hidden nodes a complex problem for which the solution is completely empirical. If there are more hidden nodes, there are more local minima, which can hamper convergence. On the other hand, having fewer hidden nodes improves the generalization of the network but allows for a less accurate fit [21]. We consider more precisely the determination of the number of hidden neurons and hidden layers in Chapters 3 and 4. The back propagation algorithm has emerged as the most popular algorithm for the supervised training of MLP. Basically it is a gradient technique and not an optimization technique. It is simple to compute locally and perform stochastic gradient descent in weight space for pattern-by-pattern updating of synaptic weights. A learning algorithm is efficient when its computational complexity is polynomial in the number of adjustable parameters that are updated from one iteration to the next. The back-propagation algorithm is also robust according to the LMS properties. On the other hand, the back-propagation algorithm could get stuck in local minima where every small change in synaptic weights increases the cost function, which impacts the performance of this method. Moreover, the rate of convergence in back-

propagation learning tends to be relatively slow, which in turn will make it computationally excruciating. In figure 2-8 three minimum points are marked.

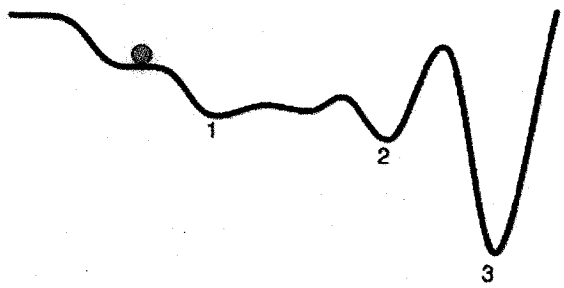

Figure 2-8: Local and global minima

Numbers one and two are the local minima of the function, number three is the global minimum of the function; back-propagation learning algorithm iteration stops as soon as the process is stuck in one of those local minima. In Chapters 3 and 4, we will consider improvement methods to overcome some of the back-propagation algorithm's drawbacks.

CHAPTER 3 PROBLEM DEFINITION AND TREATMENT OF THE DATA

Chapter 3 is divided into three sections. In Section 3.1 we define the objective of the problem and we consider the factors that participate in the regression model. In Section 3.2 we investigate the importance of the data treatment before feeding the network. In order to improve the efficiency of the applied model, we propose some data improvements; to do so, we consider two methods of normalization which are outlier elimination and distribution fitting.

3.1 Problem definition

In railroad transportation, many factors affect the number of bookings and cancellations and a lot of information is available about passengers for each departure time. In this project we aim to predict the number of passengers for a major railroad transportation company. We have decided to select the number of bookings and cancellations as the response variables of our forecasting model because the difference between these amounts represents the number of passengers at departure time. The reservation process starts 120 days before departure and there are 20 booking interval segments. During these 120 days, passengers register the information for the itinerary reservation. Some of the reservations may be cancelled through this time. There are several factors which affect number of bookings and cancellations. In this case, we investigate the impact of seven factors on the number of passengers at departure time.

This transport organization offers many departures everyday in different hours. Depending on *departure date* and *departure time* the demand differs. For example during each week there are lots of business travelers; therefore demand increases. During weekend there are some fluctuations in the number of passengers. Thus, we preferred to present departure date in two codes one for weekdays and two for

weekends. Moreover, demand changes during a day; for instance, the number of demand from 6h00 a.m. to 8h00 p.m. (i.e. during the day) is lighter than after 10h00 p.m. Therefore, the number of bookings and cancellations is affected by departure time. Hence, we express departure time in three codes. Code 1 represents the departures which are scheduled in the morning, 2 for the afternoon and 3 shows departures during the night. The first question which crosses our mind is; demand for which *product*? In transportation context, demand is considered for the tickets. In our case these tickets (i.e. products) are provided in 19 types which indicate the category of each passenger. Passengers can be student or employee, junior or senior, handicapped or VIP and so on. Depending on different products demand differs. These nineteen kinds of products are indicated as codes from one to nineteen. Tickets are allocated in fourteen types of *classes*. According to each class the price of tickets and consequently demand is different. For example, tickets in business classes are more expensive than in economic classes on account of providing a lot of facilities. Thus, we expect people usually tend to buy economic class tickets. Therefore, this is an important factor which has influence on the number of passengers. Also in this case, all types of classes are determined by codes as one to fourteen.

The reservations are affected by different prices such as, *class*, *itinerary*, and *day average price*. Depending on each class, and departure date the average prices are different. Due to its price, the demand for business class is normally less than the demand for economic class. The time of day and the day of the year also affect the price and corresponding demand of each itinerary. In contrast, these three variables are positive real values. The seven inputs of the regression model are presented in Table 3-1.

Table 3-1: Table of inputs

Name of inputs	Data type	Data range	Data description
Departure date	Code	{1,2}	Weekday, weekend
Departure time	Code	{1,2,3}	[6h25-11h55] [12h25-17h55] [18h25-21h55]
Product	Code	19 groups	Table3-2
Class	Code	14 groups	Table 3-3
Class average price	Real Value	Maximum=117.5 Minimum=24.5	--------
Itinerary average price	Real Value	Maximum=117.5 Minimum=70.86	--------
Day average price	Real Value	Maximum=93.61 Minimum=78.49	--------

So far, our artificial neural network model has seven inputs and two outputs to be predicted.

3.2 Treatment of the data

Prediction models deal with data in a mechanical way; that is, they expect the data to be specified in a standard form. Given this form, they perform a comprehensive search of the possibilities. While prediction methods may have very strong theoretical capabilities, in practice they may be limited by the shortage of data relative to the unlimited space of possibilities that they may search. This is true even for large datasets. Prediction methods will benefit from any insight into the problem that leads to a revised and improved set of features. Data analysis organizes data into a standard form that is ready for processing by prediction programs and prepares features that lead to the best predictive performance. To find the most efficient format, it is important to examine the types of features that fit the model in order to increase predictive performance. They must be encoded in a numerical format. The measured values can be scaled to a specified range for example, -1, 0, and 1.

In the remainder of this section, we investigate improvements to enhance the performance and generalization of the network including removing distortions and data normalization.

As we introduced the predictors of the model in the preceding section, we find out that departure date, departure time, products and classes are determined by codes. That is, discrete numbers which are named to each category and they do not follow a particular distribution. Thus, we enter these data into the network without any transformation. The process of providing a forecasting model in a multi-layer perceptron is divided into three phases. The first one is *training* in which the unknown parameters are fixed through iterations by 80% of data at hand (i.e January 2005). These parameters are verified in *testing* process by the rest of data which was used to train the network. Finally, to validate generalization capability of the model we apply one statistical method called, *cross validation*. This part of the process is done by applying data of March 2005. We will consider the more details in the next chapter. Preliminary results by taking outliers into account obtain unacceptable results and the error is too high. Thus, the data has to be understandable for the model. As the first step, we have to detect the outliers. Outliers are the information which imposes a significant noise in average and variance of the whole data. They can cause distortion in normalization and training process of the model. To do so, we have divided the information into 4 distinct intervals. The capacity of each train is limited; therefore, the number of bookings and cancellations, in a single request, for each itinerary is rarely more than 300 people. On the other hand, there are lots of departures with less than 100 passengers. The following tables present the data frequency to the outputs, that is, number of bookings and cancellations. In these tables, by range we mean the number of passengers in each departure. Nineteen groups of products and fourteen types of classes provide a lot of possibilities for each passenger to be located in.

Table 3-2: Table of frequencies of January 2005 for the number of bookings and cancellations (Training phase)

Range	Booking	Cancellation
$x = 0$	8907	10497
$0 < x < 100$	10502	9211
$100 < x < 300$	407	117
$x > 300$	10	1

Table 3-2 presents data frequencies for January 2005, which was used to train the network. We split the data into four intervals. The first column specifies the number of bookings and the second the number of cancellations. For example the first row of table 3-2 shows that for a specific departure date and time and particular combination of class and product, in 8907 cases during January of 2005 we had zero reservations. For instance, there were some cases in which there were no juniors in Business class for a specific departure date and time.

Table 3-3: Table of frequencies of January 2005 for the number of bookings and cancellations (Testing phase)

Range	Booking	Cancellation
$x = 0$	1906	2262
$0 < x < 100$	2248	1955
$100 < x < 300$	92	32
$x > 300$	3	0

Table 3-3 presents the data frequency that belongs to the data of January 2005, which we have applied to test the network.

Table 3-4: Table of frequencies of March 2005 for the number of bookings and cancellations (Cross validation phase)

Range	Booking	Cancellation
$x = 0$	19602	22125
$0 < x < 100$	19652	17441
$100 < x < 300$	531	221
$x > 300$	5	3

Table 3-4 presents the data frequency for March 2005, which we applied to implement cross validation for the network.

According to the preceding tables, we find that there is significant variance in both datasets which causes distortion in normalization and training process; it also creates noise in the performance of the network. To overcome this problem, we remove the outliers in an empirical way. As shown in Table 3-2, 3-3, and 3-4 the majority of the data is located in the two first intervals, motivating us to calculate the proportion of outliers for both bookings and cancellations for both dataset (i.e., January 2005 and March 2005). Table 3-5 represents proportion of outliers for each set of data and for each output. These outliers are defined as the proportion of the departures with more than one hundred passengers over the whole number of passengers.

Table 3-5: Proportion of outliers of output variables

	January 2005 (training, 80% of data)	January 2005 (testing, 20% of data)	March 2005 (Cross validation)
Bookings	2%	2%	1%
Cancellations	0.5%	0.7%	0.5%

The first result denotes the proportion of outliers for number of bookings for January 2005 training data. In this case, the ratio of outliers is 2%. The second result is the proportion of outliers for the number of cancellations, which for these data is 0.5%. All proportions are negligible and the outliers can be removed. After outlier elimination the

results are improved significantly, also the performance of the network is modified. The results are available in Chapter 5.

Generally, neural networks need range normalization. If the data is not normalized, the process will overweight those features that have larger values. Variables in the dataset should be normalized both across range and in distribution. There is much to be learned from examining the results of this transformation. Depending on data structure in each case the method differs; moreover, if a method normalizes training data, the identical method must be applied to all future data. Therefore, it is necessary to treat data systematically. In neural networks, the values of the original data should be normalized to numbers having absolute values of less than one. Experience has shown that transformed numbers of output values lead to better training. Normalization is determined from training data and a description of the scale factor is part of the solution. There are different ways to normalize the output values and, to varying extents, normalizations degrade the interpretably of feature values. Normalization by standard deviations often works well with distance measures, but transforms the data into a form unrecognizable from the original. A continuous function is also needed, which preserves the characteristics of the data and which also must be reversible to calculate the generalization error of the network. We apply the normalization method only for the output of the model because the inputs are mixed of real numbers and mostly in codes. Therefore, we enter this information into the network without any transformation. It is preferable to fit a probability function that determines the characteristics of the data and is also reversible. However Gaussian is the most common distribution in normalization context, it does not have the property of being reversible because two different experiences have the same probability of being happened.

In our case, we find out that the output values are always positive, since the number of bookings (i.e. the number of reservations) and cancellations (i.e. the number of reservations which are cancelled by approaching departure date) are always positive.

Therefore, we can use the distributions that provide positive values. In addition, the shape of the data is strictly decreasing in terms of each interval, for example, there are many data at point zero and as the sample approaches 100 (after eliminating the outliers), the number of bookings and cancellations decrease. This suggests selecting the exponential distribution to map the output values into the interval [0, 1), which is inversible and we calculate the inverse with logarithm distribution. The general formulation of the exponential distribution that we applied is

$$f(x)=\frac{1}{\lambda}e^{-\frac{x}{\lambda}} \qquad x\geq 0 \tag{3.1}$$

The estimated value for the parameter λ of the exponential distribution is the average of the data. The estimation of λ by applying statistical software is close to the average of the data. Therefore, we fit the exponential distribution to each set of data. For each set of data (i.e., train, test, cross validation) and for every set of outputs (i.e., bookings and cancellations) we have different exponential distributions with different parameters. The normalized data with an exponential distribution are all within the interval [0,1). Finally, we enter the elements of the output vectors in the exponential distribution; the exponentialized outputs will be obtained via the following equation:

$$f(y_{RealValue})=\frac{1}{\lambda_i}e^{-\frac{y_{realValue}}{\lambda_i}}=y_{exponentialized} \tag{3.2}$$

In (3.1), λ_i is the parameter for each dataset (i.e. either January or March 2005 for cancellations and bookings separately); for example, the number of bookings in January 2005's training phase follows an exponential distribution with parameter λ_i. The weights are adjusted during the training process while the difference between the output of the network and the exponentialized real values are minimized. Because the learning process tries to approach the vector of the network outputs and the exponentialized real values as much as it can by using the LMS algorithm, we suppose that the output of the network follows the same distribution as the real values.

Therefore, in order to compare the vector of real values and the output of the network, we retransform them and calculate the error. The absolute value of transformed output and the corresponding error function will be as follows,

$$y_{transformed} = \lambda \log(\lambda y_{NetworkOutput}) \tag{3.3}$$

$$Error = \vec{Y}_{RealValue} - \vec{Y}_{Transformed} \tag{3.4}$$

Using an exponential distribution has a significant impact on improving the performance and generalization of the network (results presented in Chapter 5). The exponential distribution is a bijective distribution and we can calculate the inverse distribution. In cases for which we cannot approximate or fit an appropriate distribution for large scale we can use the Cumulative Gaussian distribution, which is always positive.

Figure 3-1 shows exponential and Gaussian distribution. It is clear that in exponential distribution for each event there is just one probability of being happened while in normal distribution we have one probability for two events. In addition the cumulative of both functions are illustrated.

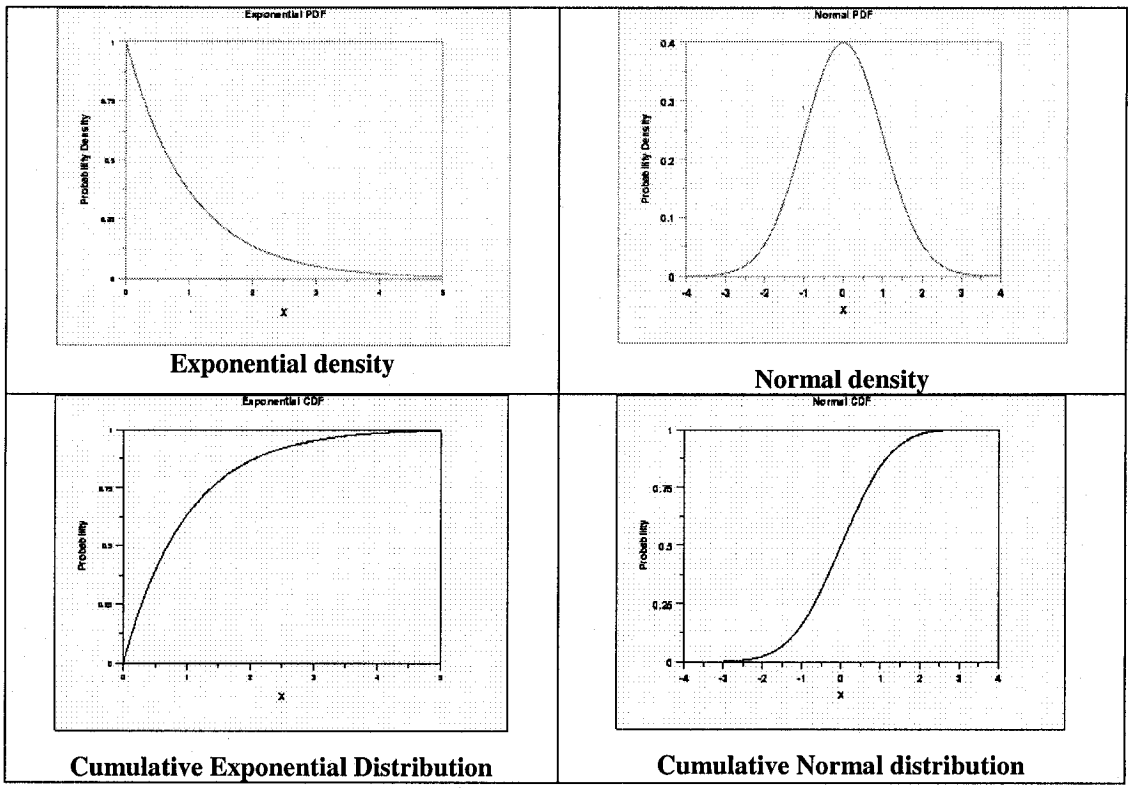

Figure 3-1: Exponential and Normal Probability and Cumulative functions

Both cumulative functions are bijective and their inverse function can be calculated.

CHAPTER 4 MODEL STRUCTURE AND IMPROVEMENTS

In this chapter, we explain the details about the structure of the MLP. Figure 4-1 illustrates the Multi-Layer Perceptron that we have applied in this case. In Chapter 3, we have investigated data structure and methods of improvement for data to reduce the error. Now, we consider the structure of the black box and the relevant methods to improve generalization of the network.

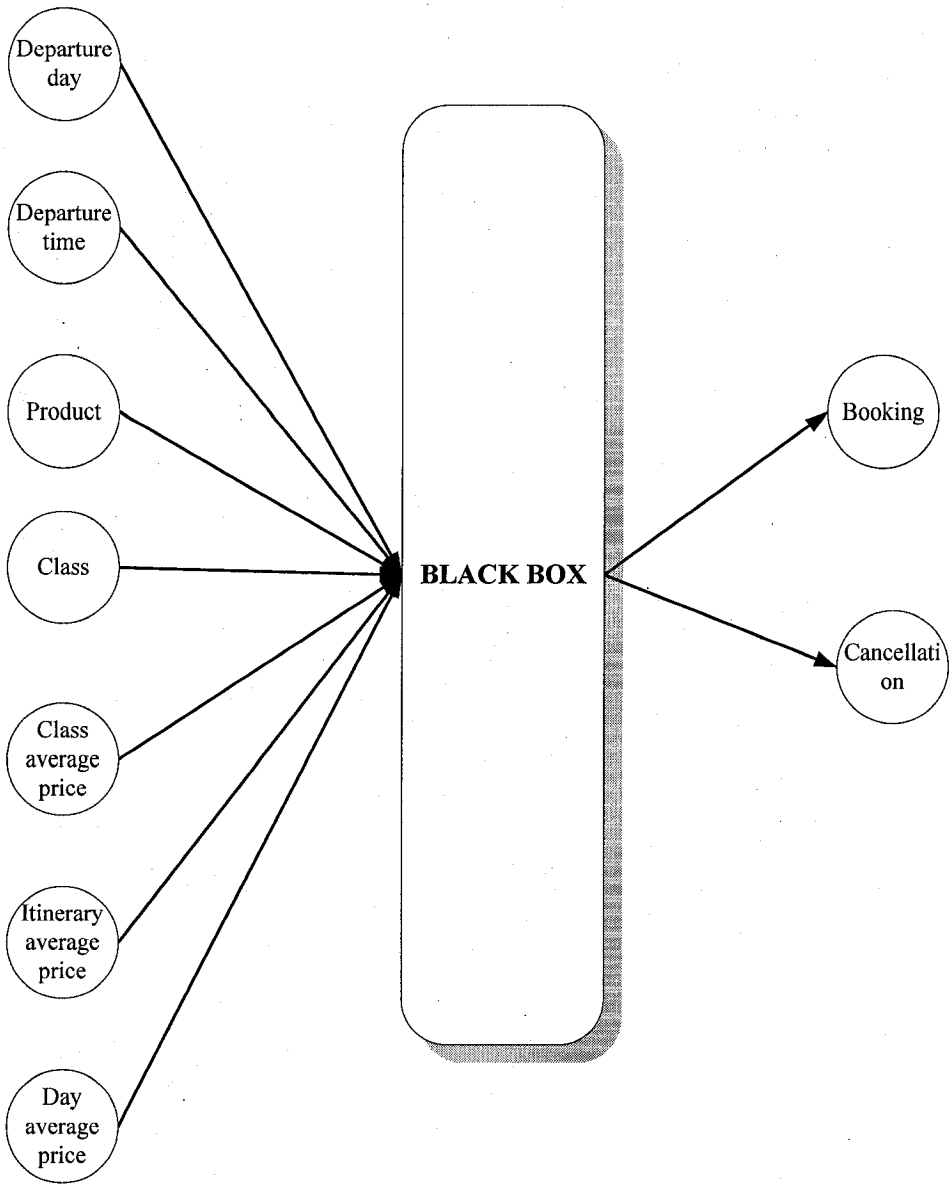

Figure 4-1: MLP Structure

4.1 Characteristics of the MLP

In this section we investigate the architecture of the MLP in use. According to the Cover theorem, one multilayer perceptron having one hidden layer with sufficient hidden neurons is enough to forecast every nonlinear relational function [22]. Choosing the *number of hidden layers and hidden neurons* is done empirically and there is no specific

rule to do it. A practical issue that arises in this context is that of minimizing the size of the network while keeping good performance. A neural network of minimum size is less likely to introduce noise into the training data and may result in better generalization. On the other hand, a large number of hidden neurons can maximize mimic of the network and make it more complex at the same time. Therefore, to respect both requirements we have to find an appropriate number of hidden neurons. We could use one of two methods: network growing or network pruning. In network growing, we start with a small architecture for the MLP and step by step we augment the number of neurons. In network pruning method, we begin with a large structure for the MLP and eliminate the neurons as long as the error decreases. In this study, we have chosen the network growing method, in which case we start with a small MLP and then add a new neuron or a new layer of hidden neurons. To reach the appropriate number of hidden neurons in each hidden layer, we start with two hidden neurons. Preliminary results show that increasing the number of hidden neurons in the first hidden layer cannot reduce the error significantly. Thus, we add another hidden layer. According to an empirical method, at last, we stop the growing network process at the point of two hidden layers with 5 neurons in each. Simulation results and sensitivity analysis are presented in Chapter 5.

One complete presentation of the entire training set during the learning process is called an *epoch*. Determination of this parameter is empirical. After testing different trials we have chosen 300 epochs to adjust the parameters during the network training. The reason of selecting this number is that, after 300 iterations the performance of the network mostly remains constant and we cannot see any significant decrease in the error during learning process. The learning process is maintained on an epoch-by-epoch basis until the synaptic weights of the network stabilize and the average squared error over the entire training set converges to some minimum value. We have also chosen *batch-mode* back-propagation learning, where weight updating is presented after entering all the training examples that constitute an epoch. An alternative to batch-mode is sequential mode, in which weight updating is performed after the presentation of each training

example. The stochastic nature of the sequential mode makes it difficult to establish theoretical conditions for convergence of the algorithm; in contrast, batch-mode provides an accurate estimate of the gradient vector and convergence to a local minimum is thereby guaranteed under simple conditions. Also, the composition of the batch-mode makes it easier to parallelize than is the sequential mode and it follows the gradient more precisely.

4.2 Model improvements

Multi-Layer Perceptrons are trained through the back-propagation learning algorithm as we described in Chapter 2. This algorithm is the most popular method of training; however, it has some drawbacks which motivate us to use some techniques to reduce the error of forecasting. For instance, the *rate of convergence* in back-propagation learning tends to be relatively slow. The local convergence rates of the back-propagation algorithm are slow. In addition, another characteristic of the error surface that impacts the efficiency of the back-propagation algorithm is the presence of *local minima* scattered around the global minima. Since back-propagation learning is basically a hill climbing technique, it runs the risk of being trapped in a local minimum where every small change in synaptic weights increases the cost function. To overcome these problems and avoid their impacts on performance of the network, we applied *momentum* and *regularization* to improve the efficiency of the network. We improve the performance of the network by applying the first method which is momentum.

The back-propagation algorithm provides an approximation to the trajectory in weight space computed by the method of steepest descent. The smaller we make the learning rate parameter η, the smaller the changes to the synaptic weights in the network will be from one iteration to the next and the smoother the trajectory will be in weight space. This improvement, however, is attained at the cost of a slower rate of learning. On the other hand, if we make η large in order to speed up the rate of learning, the resulting large changes in the synaptic weights assume such a form that the network may become

unstable. A simple method of increasing the rate of learning yet avoiding the danger of instability is to modify the delta rule by including a *momentum term* as

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n) \tag{4.1}$$

where α is usually a positive number called the momentum constant. This controls the feedback loop acting around $\Delta w_{ji}(n)$. In order to see the effect of the sequence of the pattern presentations on the synaptic weights due to the momentum constant α we write a time series with index t. The index t goes from the initial time 0 to the current time n. We have,

$$\Delta w_{ji}(n) = \eta \sum_{t=0}^{n} \alpha^{n-t} \delta_j(t) y_i(t) \tag{4.2}$$

which represents a time series of length $n+1$. Using a multilayer perceptron with two hidden layers, we tried different combinations of learning rates and momentum constants to observe their effects on network convergence. After repeating different trials through the epochs we find out that 0.9 is the most appropriate value as the momentum constant.

Another method to modify the performance of the network is *regularization*, or modifying the performance function, which is normally chosen to be the sum of squares of the network errors on the training set, by adding some fraction of the squares of the network weights. The typical performance function used for training feed-forward neural networks is the mean sum of squares of the network errors [22].

$$F = mse = \frac{1}{N} \sum_{i=1}^{N} (e_i)^2 = \frac{1}{N} \sum_{i=1}^{N} (d_i - y_i)^2 \tag{4.3}$$

We can improve generalization by modifying the performance function by adding a term that consists of the mean of the sum of squares of the network weights and biases:

$$msereg = \gamma mse + (1-\gamma) msw \tag{4.4}$$

where γ is the performance ratio, and

$$msw = \frac{1}{n}\sum_{j=1}^{n} w_j^{\,2} \tag{4.5}$$

Using this performance function will cause the network to have smaller weights and biases, which will force the network response to be smoother and less likely to deal with overfitting problem. We also applied this method to reduce the generalization error of the network.

Back-propagation is one of the simplest optimization methods, with many desirable qualities, including simplicity and randomness of case training. The conjugate gradient method trains much faster and is also effective. In order to investigate the proper method of minimization of the error function we know if the maximum number of weights can be kept low, possibly in conjunction with feature selection, then second-order optimization methods are the most reliable training methods. However, in our case, there are a lot of data available to train the network, so we used the steepest descent method, in which the gradient can be computed locally [22].

In order to determine the parameter of learning ratio and modify the training process, we employed the *adaptive learning* method. The performance of the steepest descent algorithm can be improved if we allow the parameter to change during the training process. An adaptive learning rate will attempt to keep the step size as large as possible while keeping training process stable. This parameter is made responsive to the complexity of the local error surface and it requires some changes in the training procedure. First, the initial network output and error are calculated. At each epoch, new weights and biases are calculated using the current parameter. New outputs and errors are then calculated. As with momentum, if the new error exceeds the old error by more than a predefined ratio, the new weights and biases are discarded and the learning rate is decreased; otherwise, the new weights are kept. If the new error is less than the old error, then the parameter is increased. This procedure increases the learning rate, but only to the extent that the network can learn without large error increases. Thus, a near optimal value is obtained for the local terrain. When a larger ratio could result in stable

learning, it is increased whereas the learning rate is too high to guarantee a decrease in error; it gets decreased until stable learning resumes [22].

4.3 Sigmoid Function

Now, we investigate the other components of the MLP. Regression methods try to approximate and generalize data with curves that smooth out random variations. Data smoothing can be understood as doing the same kind of smoothing on the features themselves with the objective of removing noise. From the perspective of generalization to the new cases, even features that are expected to have little error in their values may benefit from smoothing their value to reduce random variation. The primary focus of regression methods is to smooth the predicted output variable, but complex regression smoothing cannot be done for every feature. Neural networks with *sigmoid functions* that use the mean value of a partition have smoothers implicit in their representation. The sigmoid function, whose graph is S-shaped, is by far the most common form of activation function used in the construction of artificial neural networks and it is differentiable. It is defined as a strictly increasing function that exhibits a graceful balance between linear and nonlinear behavior. An example of the sigmoid function is as follows,

$$\varphi(x) = \frac{1}{1+e^{-ax}} \tag{4.6}$$

where a is the *slope parameter* of the sigmoid function. By varying the parameter a, we obtained sigmoid functions of different slopes. Figure 4-2 shows sigmoid function with different values of a.

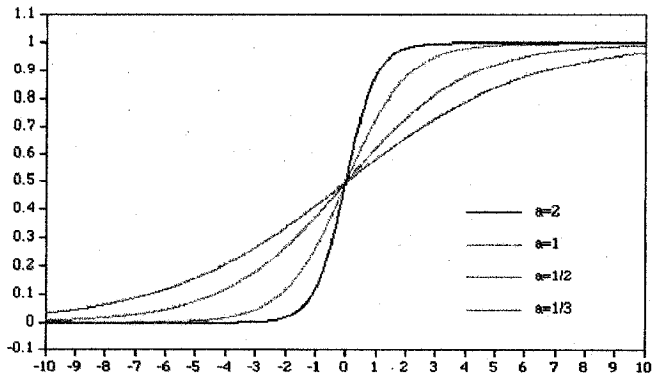

Figure 4-2: Sigmoid function

When a is small, the network needs more data to be trained and when it is large, the generalization of the network is not good enough. In our study, According to the sensitivity analysis, we have chosen $a=1$ empirically. The reason is that by choosing 1 for the parameter we do not need a large number of samples to train the network; moreover, the network is capable to perform the regression function properly.

4.4 Testing the network

After constructing the network with appropriate numbers of hidden layers and hidden neurons and after training the network, the parameters are fixed. The training process stops as soon as one of the stop criteria is met. To define the *stopping criteria*, we consider two methods: first, we can predefine an error of performance for the network and once the iterative process reaches this point, the process stops. In another method, the process finishes the adjustments as soon as the gradient method becomes stuck in local or global minima.

We are interested in verifying the estimated parameter by stimulating the network with a set of data. Here, it is important to choose the optimum amount of data to train and test the network. One of the most common problems in MLPs is *overfitting*. Overfitting occurs when the error on the training set is driven to a very small value, but when new data is presented to the network, the error is large. If the number of parameters in the

network is much smaller than the total number of points in the training set, then there is little or no chance of overfitting. If more data are collected and the size of the training set is increased, then there is no need to apply techniques to prevent overfitting. In addition to this method, using regularization can help to overcome this problem. The system has the ability to extrapolate responses for new data. This is an important feature since in general the system will be deployed and the performance obtained in the training set should also generalize to the new data. Unfortunately, due to the methodology utilized to derive the parameter values, one can never be exactly sure of how well the system will respond to new data. For this reason, it is important to use a test set to verify the system performance before deploying it to the real world. The test set consists of new data, but for which we still know the desired response. Normally, there will be a slight decrease in performance from the training set. If the performance in the test set is not acceptable, one has to go back to the drawing board. When this happens in regression, the most common source is a lack of data in training or an unsatisfactory coverage of experimental conditions [22].

In order to test the network, we use the same data set as training. We apply the January 2005 data during both the testing and training phases, but to deal with the overfitting problem, we use 80% of the data randomly to train the network and the other 20% to test the fixed parameters. This percentage is chosen empirically while it is the most common proportion for training and testing set.

4.5 Evaluation of network generalization and cross validation

Generalization is influenced by three factors: (1) the size of the training set and how representative it is of the environment of the interest; (2) the architecture of the neural network; and (3) the physical complexity of the problem at hand. To examine the network's generalizing ability we use cross validation. Cross-validation, sometimes called rotation estimation, is the statistical practice of partitioning a sample of data into subsets such that the analysis is initially performed on a single subset, while the other subset(s) are retained for subsequent use in confirming and validating the initial

analysis. The motivation here is to validate the model on a different dataset than the one used for parameter estimation. There is, however, the possibility that the model with the best-performing parameter values may end up overfitting the validation subset. In this survey, we apply different kinds of cross validations to ensure the generalization of the network, and we use multifold cross validation by dividing the set into K subsets. The model is trained on all but one of the subsets and the validation error is measured by testing it on the remaining. This procedure is repeated for a total of K trials, each time using a different subset for validation. The performance of the model is assessed by averaging the squared error under validation over all the trials of the experiment. If K gets too small, the error estimate is pessimistically biased because of the difference in training-set size between the full-sample analysis and the cross-validation analyses. In contrast, if K is too large, it may require an excessive amount of computation since the model has to be trained K times with $1 < K \leq N$ where N is the number of examples. A value of 5 or 10 for K is popular for estimating generalization error. Values of K work even better if several different random K-way splits of the data are analyzed to reduce the variability of the cross-validation estimate, even if they are as small as 5. However, in many cases, five or ten folds have been chosen to do the multifold cross validation, as our sensitivity analysis indicated that five folds is an adequate number. The results are presented in the next chapter.

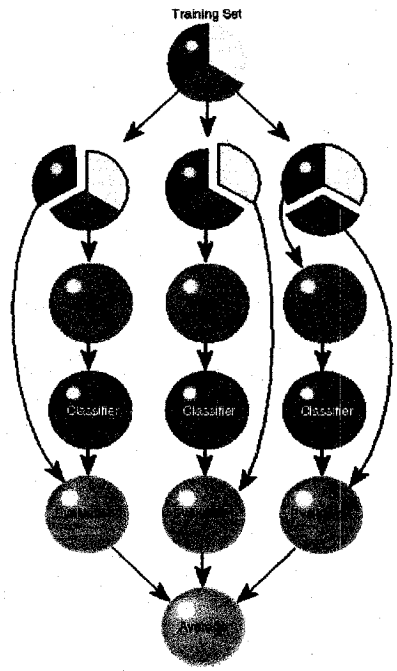

Figure 4-3: An example for 3-Fold Cross Validation

We used a second method of cross validation to examine the generalization ability of the network. A stopped network is tested on an independent dataset that has not been used for training to give an unbiased estimate on the network performance. We trained the network on a randomly chosen a subset of January 2005 for learning and validated the network with the March 2005 data. In the next chapter, we consider the results obtained by applying the model described in this chapter.

In order to construct the model and employ the methods of improvement, we have applied MATLAB software which is very powerful software in Neural Networks. There are lots of properties to use different method of optimization for error reduction and also to employ the Back-propagation learning algorithm. In the part of data treatment to find the appropriate exponential distribution we have used STATISTICA.

CHAPTER 5 TRAINING AND TESTING

In this chapter we present the training process and network performance, including the network behavior before and after improvement methods. The results of the cross-validation to verify the network capability in generalizing are also presented.

5.1 Data

In network training, outliers affect the generalization error due to the influence of these values on the mapping process; moreover, they introduce noise into normalization. Outlier elimination is one of the improvement methods that we have applied in order to reduce the forecasting error. As mentioned before, the data should be normalized before entering the network. This process is done according to the data structure. As discussed before, the exponential distribution is chosen as the most appropriate distribution in order to normalize the data before feeding the network. In the following figures, the normalization process of three main sets (i.e., training, testing, cross validation) for both bookings and cancellations is presented. Figures 5-1 and 5-2 show the exponential fit for the data that were used to train the network; we consider bookings and cancellations in two separate graphs.

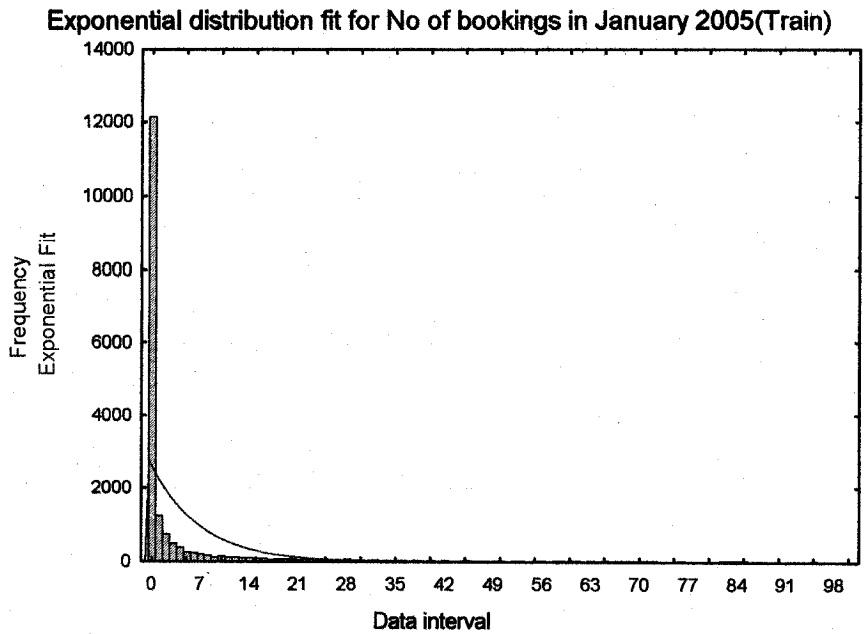

Figure 5-1: Exponential distribution fit (January 2005) for bookings

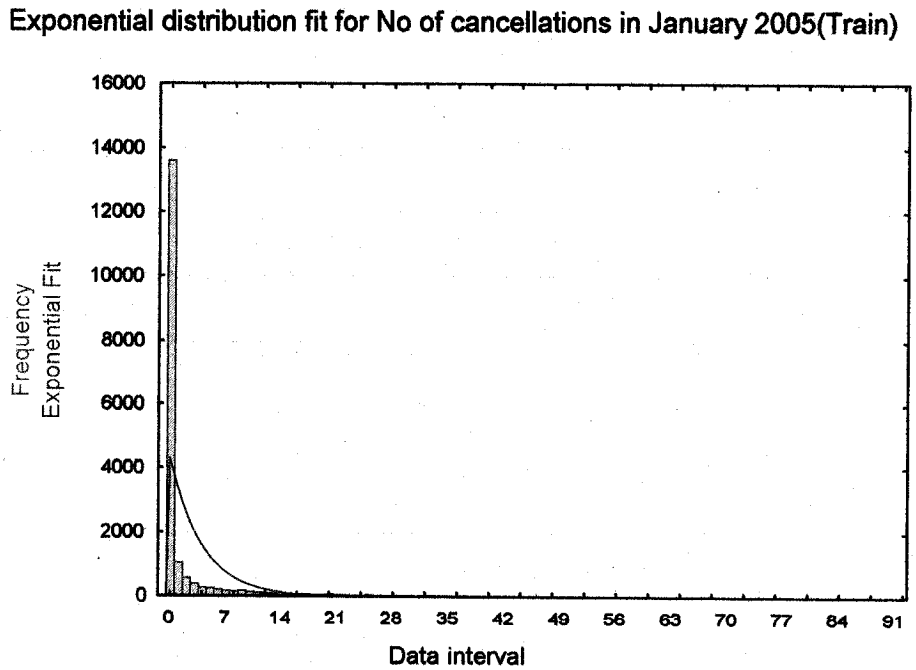

Figure 5-2: Exponential distribution fit (January 2005) for cancellations

The following two graphs show the results for the test performed on the January 2005 dataset.

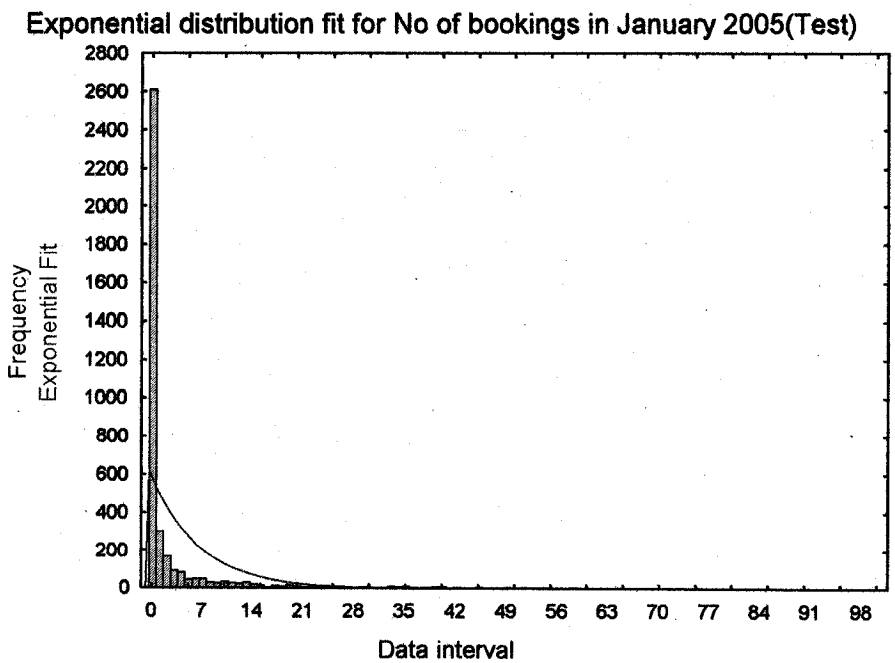

Figure 5-3: Exponential distribution fit (January 2005) for bookings

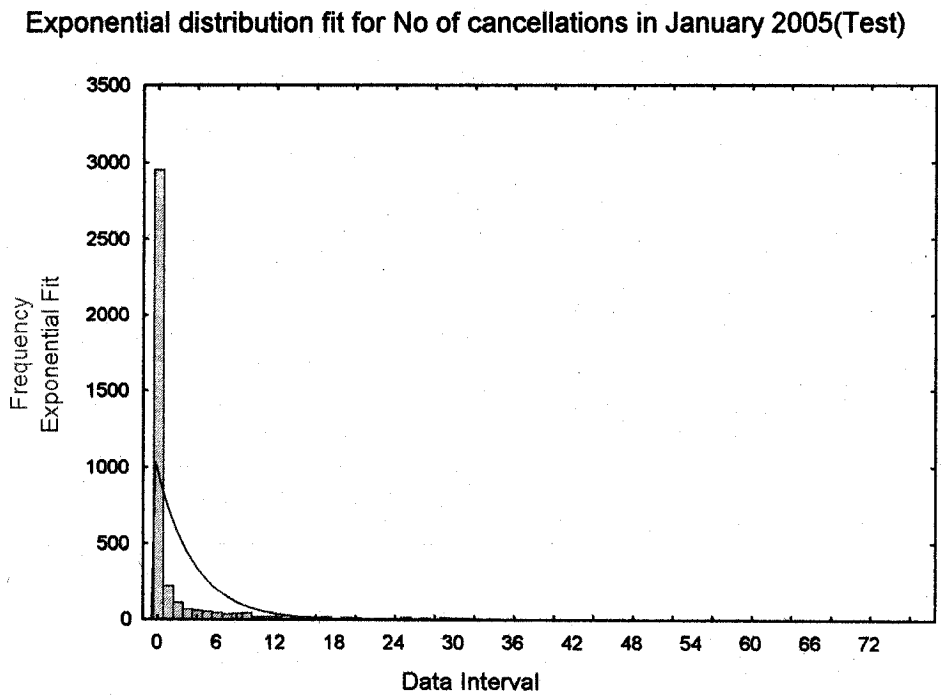

Figure 5-4: Exponential distribution fit (January 2005) for cancellations

In the following sections, we evaluate the network generalization via cross validation methods. One of the applied approaches to validate the network is done with the aid of a new dataset, for which we used the data from March 2005. We must also implement the same normalization approach for this dataset. Figures 5-5 and 5-6 illustrate the results for bookings and cancellations of the cross validation dataset.

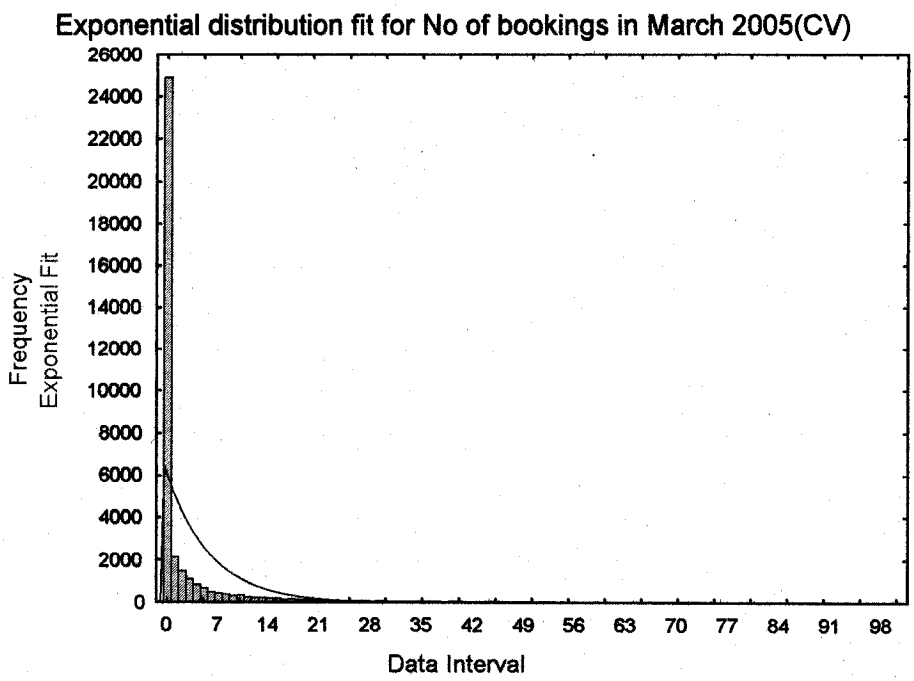

Figure 5-5: Exponential distribution fit (March 2005) for bookings

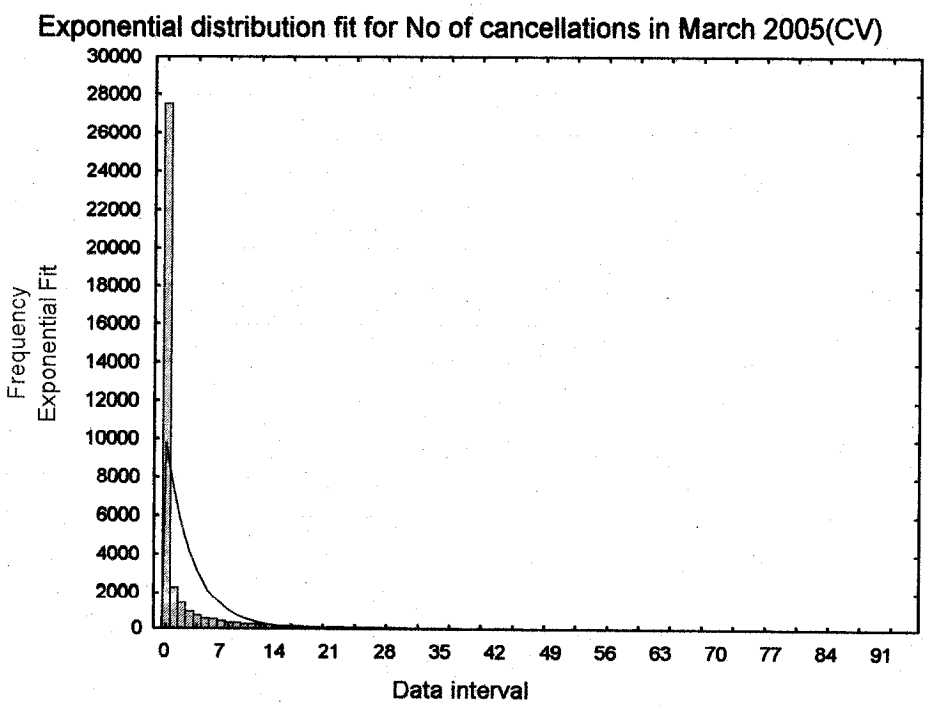

Figure 5-6: Exponential distribution fit (March 2005) for cancellations

5.2 Model

The goal of developing this network is to forecast the number of passengers at departure time and, consequently, have a regression model with minimum error of prediction. In the following graph, the number of passengers for a sample consisting of 100 observations (i.e. we represent the number of passengers for different departure times and departure days in different classes; each case is indicated by an observation in horizontal axis) is illustrated. By number of passengers, we mean the number of bookings minus number of cancellations at departure time. We expect that our model will estimate as close as possible these values.

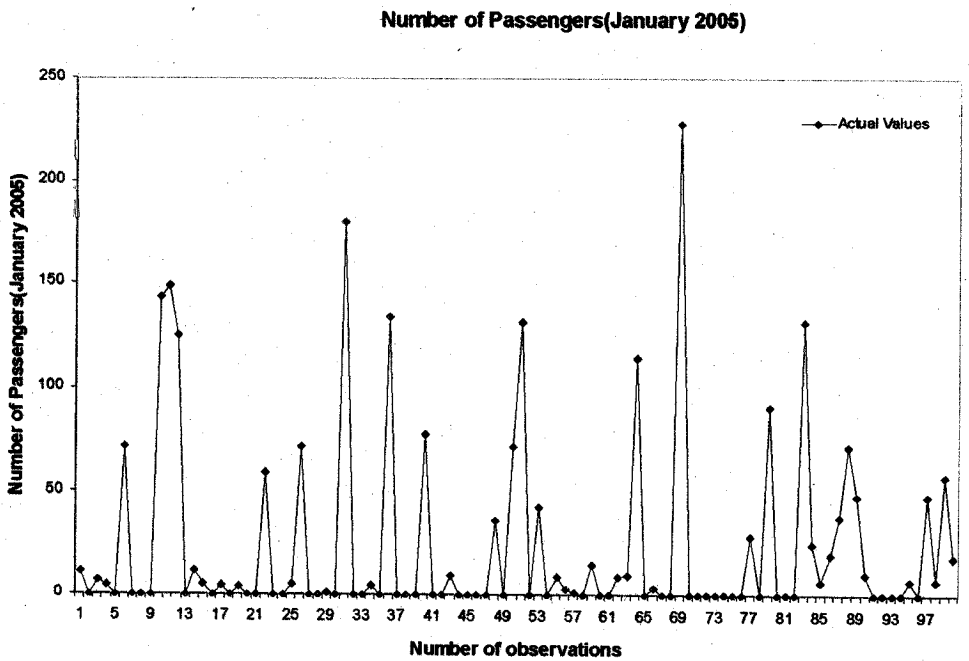

Figure 5-7: The number of passengers at departure time in January 2005 for 100 samples

In this graph, by number of observations we mean the number of passengers for specific departure date and time for a particular combination of class and product.

Number of hidden layers and neurons

As stated before, there is no specific rule to determine the exact number of hidden layers and neurons for the network's architecture. To find out how many hidden layers and

neurons that can possibly provide better performance in terms of least error of forecasting, we start from a network with one hidden layer in which there are two hidden neurons; by increasing the number of hidden nodes, we consider the performance error of the network. This method is completely empirical and we continue the process as long as the minimum possible error is achieved. As shown in Figure 5-8, the error does not reduce significantly when the number of neurons increases. The minimum error has been obtained for five hidden neurons in the first layer. The minimum error obtained by using just one hidden layer remains more than 15%, which motivates us to add another hidden layer to see if we can decrease the error empirically. Regardless of the first layer results, we start to examine different trials while existing two hidden layers in the network architecture. Our preliminary experiments have showed that two hidden-layers have performed better prediction results rather than single hidden-layer.

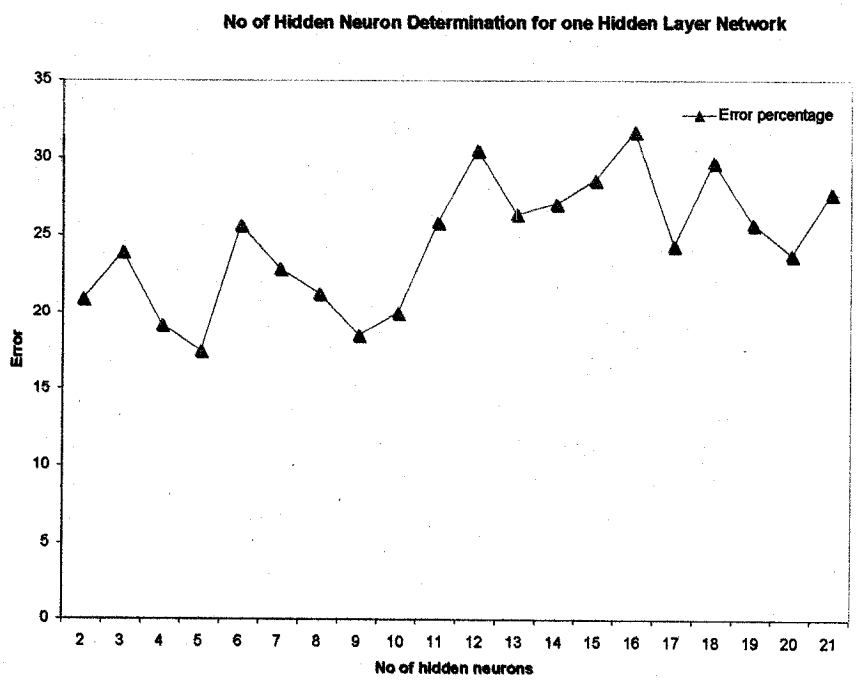

Figure 5-8: Number of hidden layers and neurons determination

We start with two hidden neurons in each layer and try all the possibilities up to seven hidden neurons in both layers as show in Figure 5-9. The experiment shows that the minimum error occurs with five neurons in each hidden layer and after that, the error gradually increases. Therefore, we construct a network with two hidden layers with five hidden neurons each.

Sigmoid transfer function

In both hidden layers, sigmoid transfer functions were used. The slope parameter of this S-shaped function could affect the network performance. If this parameter is large, it approaches the threshold function and less information is required to train the network. Technically, as long as the network is trained by more information, the possibility of overfitting (it happens whenever the performance error is low but the generalization error is high) will decrease. In contrast, if this parameter is too small, the network will need more training data and the training process will be more time consuming. After examining different values for the slope parameter, i.e. $a = 2, a = 1, a = 1/2, a = 1/3$, in terms of training time and lower generalization error (which slope parameter indirectly affects this error) we have chosen $a = 1$ as the slope parameter.

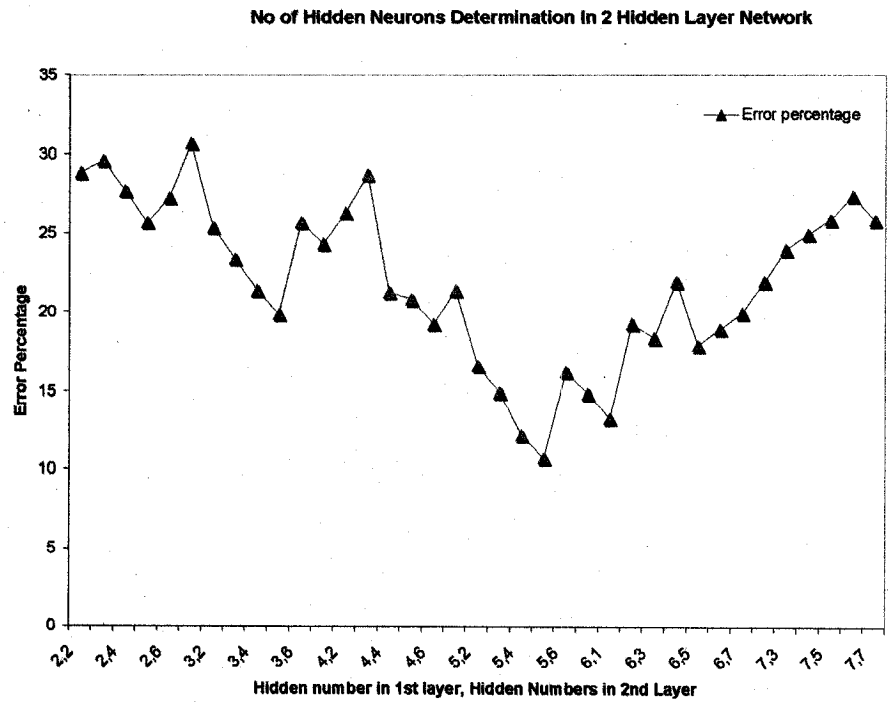

Figure 5-9: Number of hidden layers and neurons for two-layer network

Stopping criteria

In the back-propagation learning method, the network is trained using an iteration process for which the stopping criterion is either i) when the error of performance reaches the predefined error value or ii) when the line search method, in this case the steepest descent, is stuck in a local minima. The weights coefficients (i.e. coefficients of each neuron at each layer) have been initialized using the most common weight initialization random function which consists in generating random values between -1 and +1. Figures 5-10 and 5-11 depict the training process of the network in two experiments with different predefined performance goals (i.e. for the first one the predefined error is 10^{-6} and for the second one is 10^{-4}). In both cases, the training process starts naturally with large error and, during the adjustments, the error decreases gradually. The fluctuations during iterations are due to the gradient method. The training

process does not reach the predefined performance error or get stuck in a local or global minimum after 300 iterations in either case.

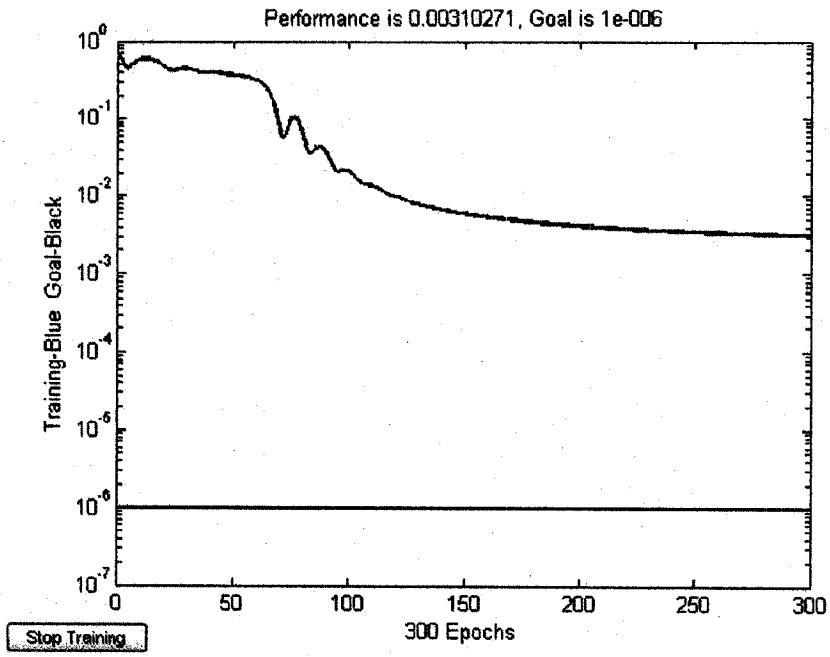

Figure 5-10: Performance process

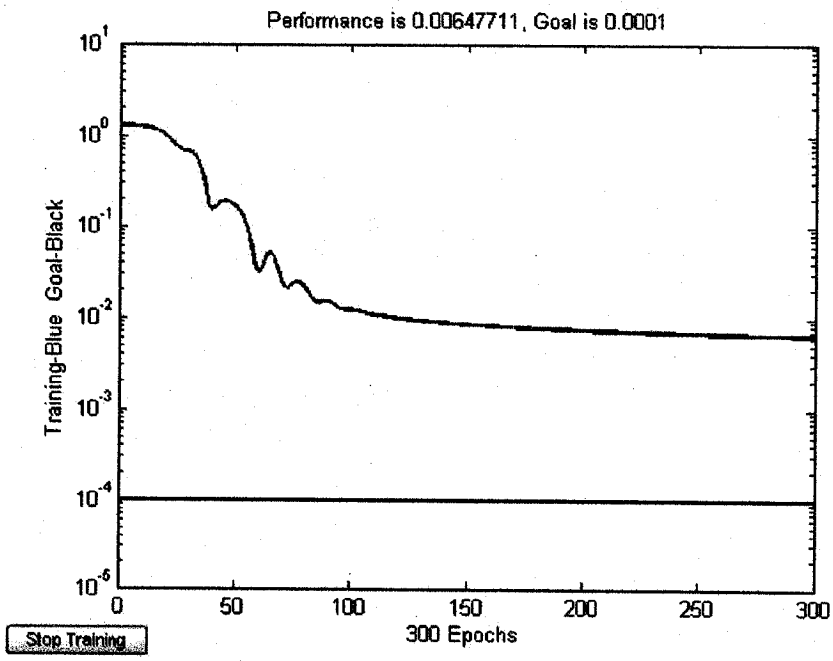

Figure 5-11: Performance process

The final structure of the neural network is illustrated in Figure 5-12.

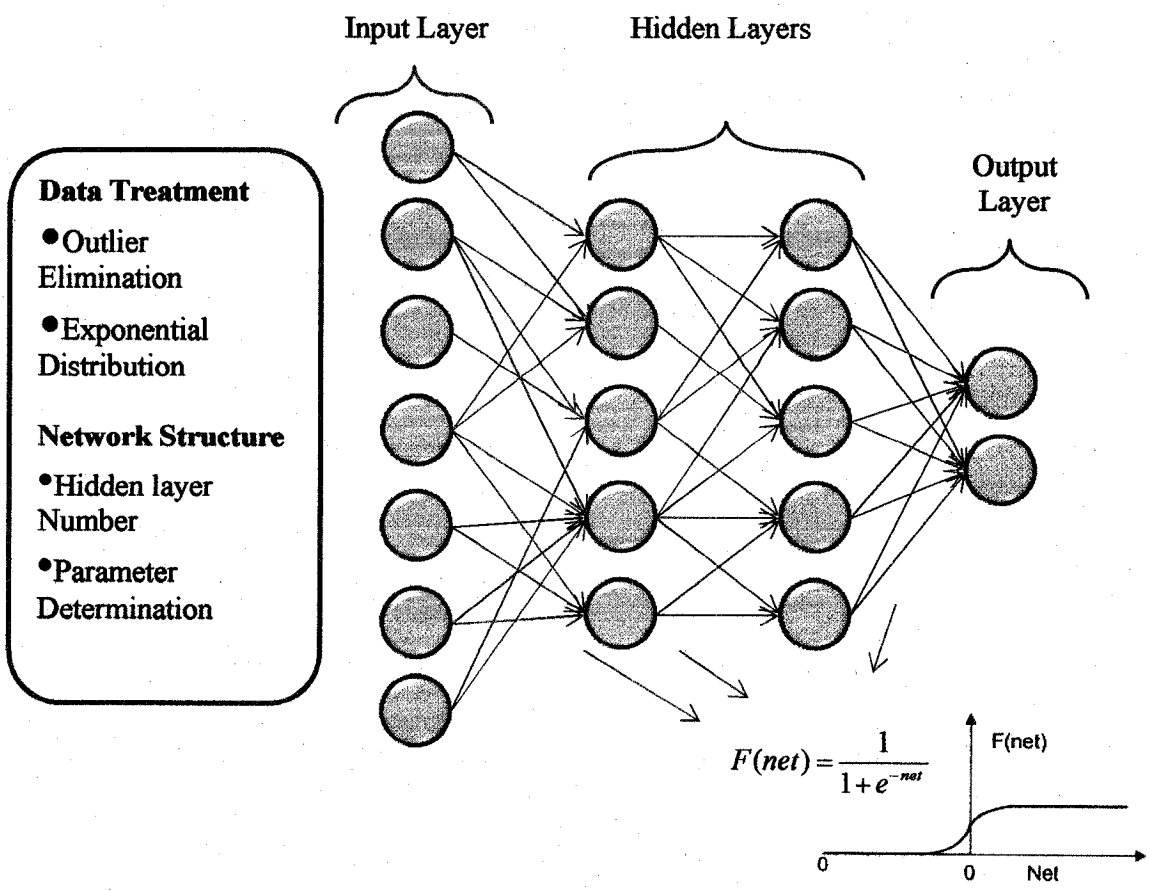

Figure 5-12: Final structure of the network

5.3 Results

In the regression process, the aim is to reduce the difference between the actual and predicted values. In this case, the predicted values are network outputs and the actual values are the numbers which were extracted from the transportation network. The preliminary results, without our improvements, are shown in Figure 5-13. The results are clearly unsatisfactory because there are significant differences between the network

outputs and the real values, making it necessary to employ some modification methods to improve the network's forecasting ability.

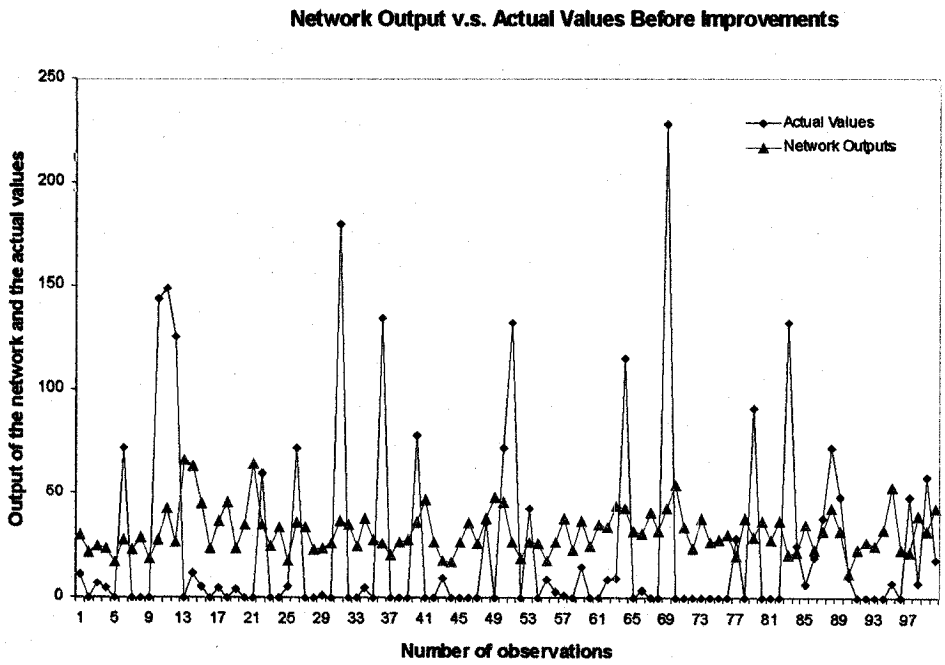

Figure 5-13: Prediction accuracy before imposing improvement methods

As discussed in the previous chapter, some improvement methods were implemented to improve the results of the network. Before applying these methods, the error was in the 35%–45% interval, but after using exponential distribution, the results have improved dramatically giving an error rate lower than 25%. After applying the adaptive learning method to control the learning rate, the results are more stable and acceptable. However, our target error is about 8%–10%, and we are still far from this result. We used momentum and regularization to reduce the error and finally, reach our target by removing outliers.

Figure 5-14 illustrates the improvements, in term of errors, obtained by these techniques. As we can see, the residuals have been reduced significantly and the network is capable of developing almost the same format as the actual values.

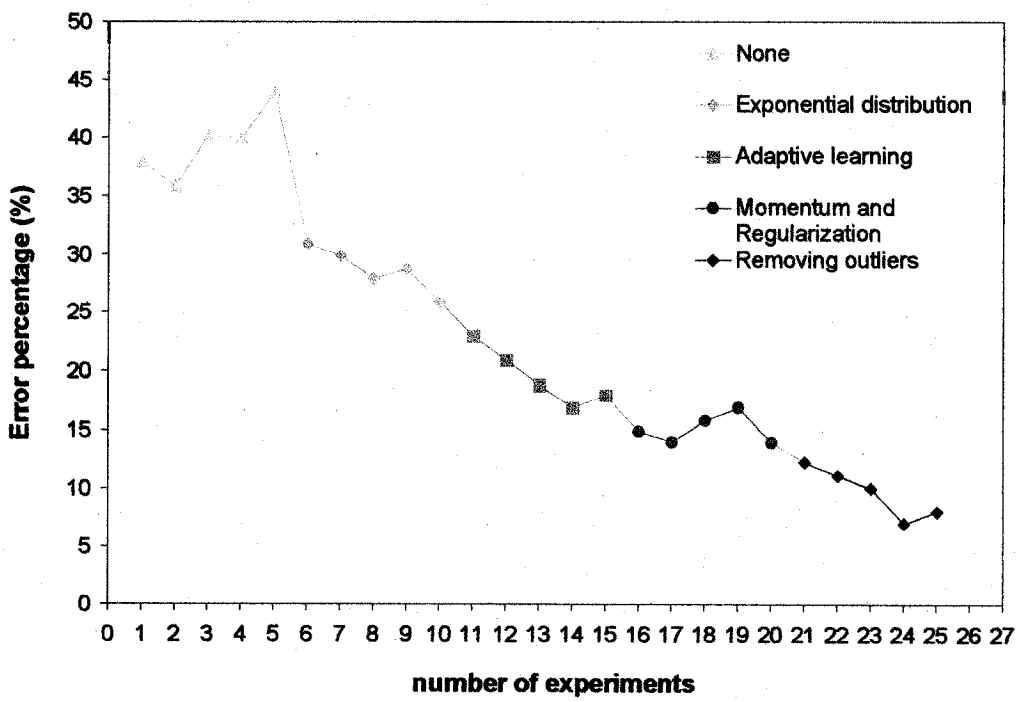

Figure 5-14: Improvements impact error reduction

Figure 5-15 illustrates the results obtained with the improvements. The figure shows that the network can reproduce, with accuracy, the actual data. Note that this figure should be contrasted with Figure 5.7.

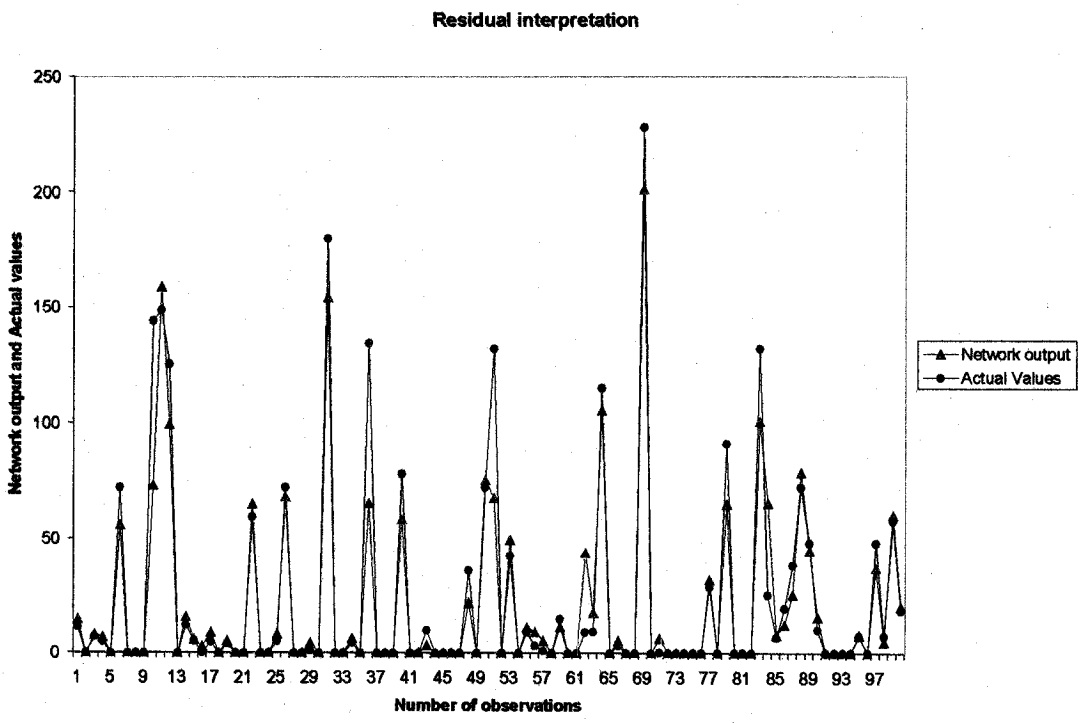

Figure 5-15: Prediction accuracy after imposing improvement methods

Generalization

After developing the multi-layer perceptron with the structure illustrated in Figure 5-12 and after applying the improvement, we expect that the network could generalize its ability of forecasting for unseen datasets as well. To evaluate network generalization, we propose two methods: K-fold cross validation (i.e. we divide data into K mutually equal sized subsets and we train and test the network K times with different trials) and using a new dataset that has not been trained by the network. First, we must decide the appropriate number of K as the number of subsets. There is no specific rule to determine K. Using 5 or 10 folds are the most common numbers for K; however, there is no certainty about this matter. Therefore, we performed a sensitivity analysis (i.e. we try different possibilities and we decide whenever more appropriate result is obtained), which leads us to start with seven folds. To represent this analysis we have tested three different possibilities in each K equals to 7, 3, and 5 respectively.

Table 5-1: 7-Fold cross validation

Number of folds	Number of incorrect estimations(Gross Bookings)	Number of incorrect estimations(Cancellations)	Gross Bookings Prediction Error	Cancellation Prediction Error
1	120	143	3.55	4.24
2	132	123	3.89	3.63
3	151	134	4.46	3.96
4	92	170	2.72	5.02
5	180	127	5.32	3.75
6	141	172	4.17	5.08
7	201	172	5.94	5.08
Average			4.29	4.53

As can be seen in Table 5-1, developing a 7-fold cross validation obtains an unrealistically low generalization error, which could cause unstable results when applied to large, new datasets. Here, we applied 86% (i.e. $6/7$) of data to train the network and used the remaining 14% (i.e. $1/7$) to test the generalization. If we apply a completely new large dataset, the result will not remain the same, so we tried a 3-fold method, which is presented in Table 5-4.

Table 5-2: 3-Fold cross validation

Number of folds	Number of incorrect estimations(Gross Bookings)	Number of incorrect estimations(Cancellations)	Gross Bookings Prediction Error	Cancellation Prediction Error
1	1027	982	13.02	12.45
2	628	826	7.96	10.47
3	923	889	11.70	11.27
Average			10.89	11.39

In 3-fold cross validation, the network could suffer from overfitting (i.e. while training the network, error rate is low; whereas, the generalization error is high because we use only 66% of the data to train the network. Also the error is high due to the lack of training. Finally, we decided to choose a 5-fold method, in which we extract 80% of data randomly to train the network and use the remaining 20% to test it. We repeated this method five times and then we calculated the average value of the runs. The results are a good representative of the generalization error of the network. Tables 5-3 and 5-4 show the results of this experiment done twice (i.e. we have repeated all trials two times to verify the variation of the average errors for both outputs; bookings and cancellations) to ensure about the stability of network generalization.

Table 5-3: 5-Fold cross validation (first test)

Number of folds	Number of incorrect estimations(Gross Bookings)	Number of incorrect estimations(Cancellations)	Gross Bookings Prediction Error	Cancellation Prediction Error
1	364	314	8.7626	7.5590
2	364	273	8.7626	6.5720
3	439	695	10.5681	16.7309
4	367	283	8.8349	6.8127
5	376	272	9.0515	6.5479
Average			9.19594	8.8445

The generalization error for booking is 9.19% and for cancellation is 8.84%. By comparing the results in the second experiment (Table 5-4), we see that there are no significant variations in the obtained generalization errors for bookings and cancellations.

Table 5-4: 5-Fold cross validation (second test)

Number of folds	Number of incorrect estimations(Gross Bookings)	Number of incorrect estimations(Cancellations)	Gross Bookings Prediction Error	Cancellation Prediction Error
1	447	432	10.7607	10.3996
2	383	272	9.2200	6.5479
3	417	272	10.0385	6.5479
4	649	337	15.6235	8.1127
5	388	289	9.3404	6.9571
Average			10.99662	7.71304

As the second method of evaluating the generalization of the network, we used a new dataset that had not been used in training process of the network, the data from March 2005. As illustrated in Figure 5-14, for 20 implementations of the network, the results are always steady in the 6%–12% interval. The fluctuations in the graph are due to the different subsets of the whole dataset that we have applied randomly. As shown in the graph, the average generalization error is almost 8%.

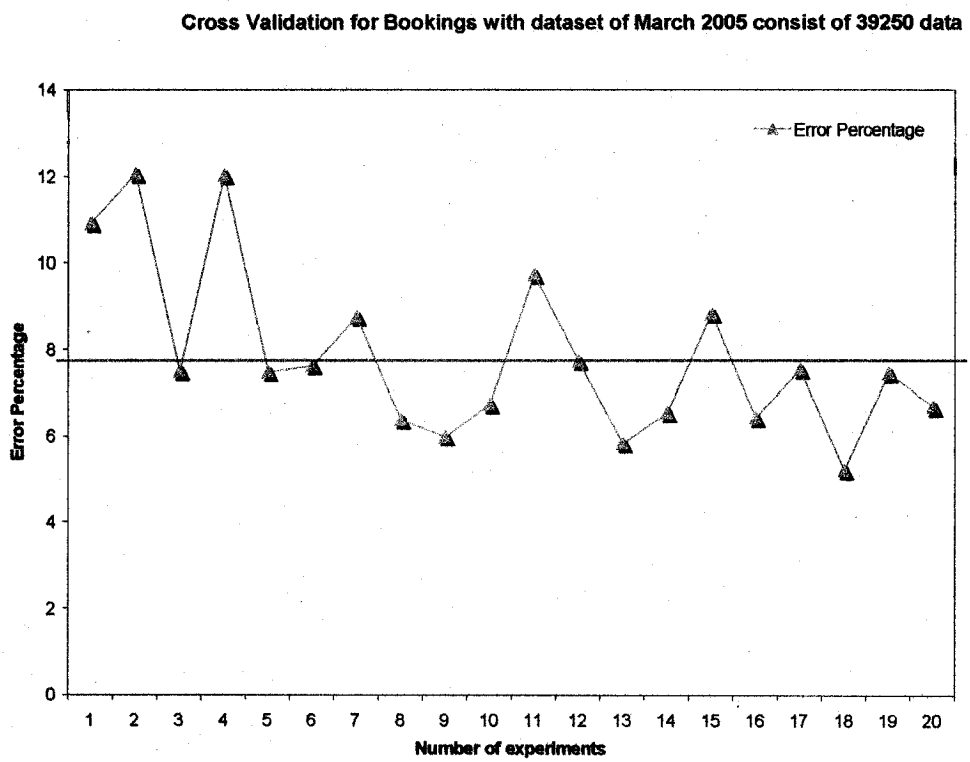

Figure 5-16: Cross validation with March 2005 data

CHAPTER 6 CONCLUSION AND DISCUSSION

Problems in transportation demand forecasting are highly nonlinear and data resources are numerous and complex. Neural networks show great promise as useful tools for analyzing these data and they are being used increasingly in industrial applications, due to the ability of neural networks to solve the non-linear regression problems successfully. This feature is a highly important aspect of neural computing, as it can be used to model a function where the data size is large and there may be some missing information.

In this study, we proposed an artificial neural network (ANN) to be used in transportation demand forecasting. The efficiency of the proposed treatment of the training dataset is validated through several experiments. It has been shown that accuracy in neural networks can be significantly improved. In this respect, the proposed technique extends our understanding of the practical aspects of neural computation and, therefore, the proposed process is applicable to any situation where neural network approaches are involved. However, the generalization of the network is affected by the different types of data structures that are used to train and test the network. That is, if the parameters are adjusted by certain kinds of data, we cannot expect consistently a low percentage of generalization error.

The fact that we have applied exponential distribution to do the treatments of data is remarkable. The parameter of the distribution could differ and it could reduce the error significantly. We have chosen the parameter which is close enough to the expected value of data. Concerning improvements of the model that have been applied to modify the performance of the feed-forward neural network and to define the structure of the ANN, we have generated hidden layers and hidden neurons as they are needed. We have started learning process with only two neurons and during learning we have added new

neurons creating a multilayer structure. The number of hidden neurons, which is the complexity of the network, increases step-by-step while the training error decreases. As a result, the training algorithm improves the neural network to a near optimal complexity that can generalize well. In order to represent the impacts of each improvement method, we kept constant all the circumstances of the network and we have interpreted the error reduction, as have been illustrated in 5-14. Each time we kept the last method and we have applied the next one to show the impact of each method individually.

Using a flexible nonlinear approximator demonstrates that it is possible to reasonably estimate an unknown function and accurately forecast the short-term behavior of a time series with complicated dataset. In neural networks, there is no strict assumption about distribution functions. The forecasting model is adaptive and can be retrained regularly to accommodate changing situations. The configuration of the neural networks (i.e., the number of input nodes and hidden nodes) can be specified based on the needs and the data availability. Neural networks are highly robust with respect to underlying data distribution and no assumptions are made about relationships between variables, such as that they are linear. Thus, neural networks possess the capability to learn nonlinear and complex relationships with limited prior knowledge of process structure. ANN models come in a variety of structures and among them the back-propagation model is most frequently used as it is based on nonlinear techniques and is capable of processing complex data. The proposed model consists of a two hidden-layer neural networks and applies the back-propagation learning algorithm to reach a generalization error for the forecasting model. This means that the network is capable of predicting future demand with only an 8% error.

However, in practice, MultiLayer Perceptrons (MLPs) have some drawbacks. They do not have the best approximation property in the class of functions on N-dimensional Euclidean space and there is much work still to be done, particularly with regard to

model interpretation and validation. This research suggests the future project dealing with improvements to the model.

According to Section 3-2, almost half of the output values of the network are zero, which could cause significant error in the performance and learning process of the network. To avoid this, we could provide another MLP or use radial basis functions to isolate these zeros and create a better configuration for the regression network. This research could be continued by applying the suggested method for model and employing data improvement.

REFFERENCES

[1] MONTGOMERY, D., JOHNSON, L., and GARDINER, J. (1990). Forecasting and time series analysis. Second Edition. McGraw-Hill (TX)

[2] ZHANG, X. (2006). Inventory control under temporal demand heteroscedasticity. *European Journal of Operational Research,* 182(1), 127-144.

[3] DEVOTO, R., FARCI, C., and LILLIU, F. (2002). Analysis and forecast of air transport demand in Saedinia's airport as a function of tourism. *Urban Transport VIII, LJ Sucharov and CA Brebbia.*

[4] CHUNG, J. H. (2002). Structural model of automobile demand in Korea. *Transportation Research Record,* 1807, 87-91.

[5] SEN, A. (1985). Examining air travel demand using time series data. *Journal of Transportation Engineering,* 111(2), 155-161.

[6] FAN, W., and CHI, H. (2004). Cluster Model for Flight Demand Forecasting. *IEEE Intelligent Control and Automation,* 4, 3170-3173.

[7] ANDERSON, M. D., SHARFI K., and GHOLSON, S. (2006). Direct demand forecasting model for small urban communities using multiple linear regression. *Transportation Research Record,* 1981, 114-117.

[8] VARAGOULI, E. G., SIMOS, T. E., and XEIDAKIS, G. S. (2005). Fitting a multiple regression line to travel demand forecasting: the case of prefecture of Xanthi, *Northern Greece. International Conference of Computational Methods in Science and Engineering*, 42(7-8), 817-836.

[9] RENGARAJU, V. R., and ARASAN, V. T. (1992). Model for air travel demand. *Journal of Transportation Engineering*, 118(3), 371-380.

[10] LIN, K. Y. (2005). Dynamic pricing with real-time demand learning. *European Journal of Operational Research*, 174(1), 522-538.

[11] GODFREY, G. A., and POWELL, W. B. (2000). Adaptive estimation of daily demands with complex calendar effects for freight transportation. *Transportation Research Part B: Methodological*, 34(6), 451-469.

[12] WIDIARTA, H., VISWANATHAN, S., and PIPLANI, R. (2006). On the effectiveness of top-down strategy for forecasting autoregressive demands. *Naval Research Logistics*, 54(2), 176-188.

[13] SNYDER, R. D., and KOEHLER, O. K. (1999). Forecasting for inventory control with exponential smoothing. *International Journal of Forecasting*, 18(1), 5-18.

[14] MILTENBURG, J., and PONG, H. C. (2006). Order quantities for style goods with two order opportunities and Bayesian updating of demand. Part I: no capacity constraint. *International Journal of Production Research*, 45(8), 1707-1723.

[15] POPOVIC, J., and TEODOROVIC, D. (1996). An adaptive method for generating demand inputs to airline seat inventory control models. *Transportation Research, Part B (Methodological)*, 31B (2), 159-178.

[16] SHARDA, R., and DELEN, D. (2005). Predicting box-office success of motion pictures with neural networks. *Expert System with Applications*, 30(2), 243-254.

[17] MOZOLIN, M., THILL, J. C., and USERY, E. L. (1999). Trip distribution forecasting with multilayer perceptron neural networks: A critical evaluation. *Transportation Research, Part B (Methodological)*, 34B (1), 53-73.

[18] HASHEMI, R., E BLANC, L., RUCKS, C., and SHEARRY, A. A. (1995). Neural Network for Transportation Safety Modeling. *Expert Systems with Applications*, 9(3), 247-256.

[19] CELIKOGLU, H. B., CIGIZOGLU, H. K. (2006). Public transportation trip flow modeling with generalized regression neural networks. Advances in Engineering Software 38(2), 71-79.

[20] BEN GHALIA, M., and WANG, P. (2000). Intelligent System to Support Judgmental Business Forecasting: The Case of Estimating Hotel Room Demand. IEEE Transactions on Fuzzy Systems, 8(4), 380-397.

[21] FENG, Y. Q., WANG, X. F., LEI, Y., and FENG, Y. J. (2002). Simulation and analysis of Chinese economic development based on neural networks. International Conference on Machine Learning and Cybernetics. 3, 1423-1428.

[22] HAYKIN, S. (1999). Neural Networks A Comprehensive Foundation. Second Edition. Upper Saddle River, NJ: Prentice Hall.